varyoff

Purpose

Removes an external disk drive from the operating system configuration.

Syntax

AJ2FL119

Description

The **varyoff** command removes an external disk drive and any minidisks residing on the disk from the existing AIX Operating System hardware and minidisk configurations. File systems residing on minidisks are unmounted and file system consistency checks are performed.

Flags

-c	Does not perform file consistency checking.
-d hdisk*num* . . .	Specifies the system device name for an external disk drive, where *num* is an integer from 0 to 33, inclusive. The corresponding drive and the minidisks residing on that drive are removed from the AIX Operating System. **unmount** and **fsck** are performed on each file system defined on the disk.
-r	Removes information about all minidisks on the external disk drive from the files **/etc/system** and **/etc/filesystems**.

Examples

1. To remove a disk from the AIX Operating System using its name:

   ```
   varyoff  -d hdisk7
   ```

   This command **unmounts** and performs **fsck** functions on each file system defined on the disk drive named `hdisk7`. `hdisk7` and all of its minidisks are removed from the operating system.

2. To remove more than one disk:

   ```
   varyoff  -d  hdisk9 hdisk7 hdisk12
   ```

   This command removes disk drives `hdisk9`, `hdisk7`, and `hdisk12` from the operating system.

3. To remove information about all minidisks on an external disk drive from the system configuration files:

   ```
   varyoff -r -d hdisk10
   ```

   This command removes information about all minidisks on the external disk drive `hdisk10` from the files **/etc/system** and **/etc/filesystems**.

4. To avoid **fsck** functions when removing a disk:

   ```
   varyoff  -c -d hdisk7
   ```

Files

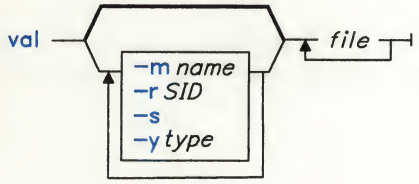
/etc/filesystems	Descriptions of mountable file systems.
/etc/system	Default system file.
/etc/system.vary.bk	System backup of **/etc/system**.
/etc/varyoff.out	Default message output file.

val

Purpose

Validates Source Code Control System (SCCS) files.

Syntax



OL805292

Description

The **val** command reads *file*s and determines if the specified *file* is an ***Source Code Control System (SCCS)*** file meeting the characteristics specified by the flags. If you specify a - (minus) for *file*, **val** reads standard input and interprets each line of standard input as **val** flags and the name of an SCCS file. **val** continues to take input until it reaches an end-of-file character **(Ctrl-D)**.

The **val** command displays error messages to standard output for each file processed. **val** also returns a single 8-bit code upon exit. The 8-bit code indicates possible mismatches or errors. It is interpreted as a bit string in which set bits (from left to right) are interpreted as follows:

bit 0 = missing file parameter
:55 1 = unknown or duplicate flag
bit 2 = damaged SCCS file
bit 3 = cannot open file or file not SCCS
bit 4 = SID is invalid
bit 5 = SID does not exist
bit 6 = %**Y**%, **-y** mismatch
bit 7 = %**M**%, **-m** mismatch

When **val** processes two or more files on a given command line or multiple command lines (when reading the standard input), a code is returned that is a logical OR of the codes generated for each command line and file processed. **val** can process up to 50 files on a single command line. Any number above 50 produces a dump.

Flags

Each flag or group of flags applies independently to each named file. The flags can appear in any order.

-m*name* Compares the value *name* with the SCCS %**M**% identification keyword in *file*. See "Identification Keywords" on page 480 for more information on the %**M**% keyword.

-r*SID* Specifies the *SID* of the *file* to be validated. The *SID* must be valid and unambiguous.

-s Suppresses the error message normally written to standard output.

-y*type* Specifies a *type* to compare with the SCCS %**Y**% identification keyword in *file*. See "Identification Keywords" on page 480 for more information on the %**Y**% keyword.

Related Information

The following commands: "**admin**" on page 41, "**delta**" on page 310, "**get**" on page 477, and "**prs**" on page 781.

The **sccsfile** file in *AIX Operating System Technical Reference.*

The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces.*

Related Information

The following commands: "**fsck, dfsck**" on page 445, "**umount, unmount**" on page 1112, "**varyon**" on page 1180, and "**vrmconfig**" on page 1206.

The discussion of external drives in *Managing the AIX Operating System.*

The **system** and **filesystems** files in *AIX Operating System Technical Reference.*

varyon

Purpose

Makes an external disk drive and any minidisks or file systems defined on it available for use.

Syntax

OL805455

Description

The **varyon** command adds an external disk drive and any minidisks or file systems residing on the disk to the existing AIX Operating System hardware and minidisk configurations. **varyon** runs the **fsck** command to perform file system consistency checks and mounts the file system if the attributes **vcheck = true** and **vmount = true** are present in the corresponding stanza of the **/etc/filesystems** file. By default, **varyon** sends only generalized completion messages to standard output, routing more detailed error messages to the file **/etc/varyon.out**.

Flags

-c	Specifies that **varyon** not perform file consistency checking.
-d hdisk*num* . . .	Specifies the system device name for an external disk drive, where *num* is an integer from 0 to 33, inclusive. The **varyon** command uses **vrmconfig** to configure minidisks residing on the drive. It uses **fsck** to check each file system whose stanza in the **/etc/filesystems** file contains the attribute **vcheck = true**. It mounts each file system that has the attribute **vmount = true**.
-q	Operates in quiet mode; does not prompt the user about minidisk names or rename the minidisks. If **/etc/system** and **/etc/filesystems** contain information about minidisks on the external disk drive, this option adds the minidisks to the system configuration using the existing minidisk names and mount directory.

Examples

1. To configure an entire external disk drive:

   ```
   varyon  -d hdisk7
   ```

   This command configures a disk drive, `hdisk7`, into the operating system, configures any minidisks defined on the disk, and performs **fsck** and **mount** functions on file systems as specified by the **/etc/filesystems** file.

2. To configure more than one external disk drive:

   ```
   varyon  -d  hdisk9 hdisk7 hdisk12
   ```

   This command makes disk drives `hdisk9`, `hdisk7`, and `hdisk12` available for use.

3. To configure an external drive without performing **fsck** functions:

   ```
   varyon  -c  -d  hdisk7  hdisk8  hdisk11
   ```

4. To make the external disk drive available without prompting the user, even if minidisks are already defined in the system configuration files:

   ```
   varyon -q -d hdisk3
   ```

   This command adds the minidisks to the system without prompting the user if **/etc/system** and **/etc/filesystems** contain information about any of the minidisks on the drive. If information exists the system uses the existing minidisk names and mount directory.

Files

/etc/filesystems	Descriptions of mountable file systems
/etc/system	Default system file
/etc/system.vary.bk	System backup of **/etc/system**
/etc/varyon.out	Default message output file

Related Information

The following commands: "**fsck**, **dfsck**" on page 445, "**mount**" on page 669, "**varyoff**" on page 1177 and "**vrmconfig**" on page 1206.

The discussion of external drives in *Managing the AIX Operating System.*

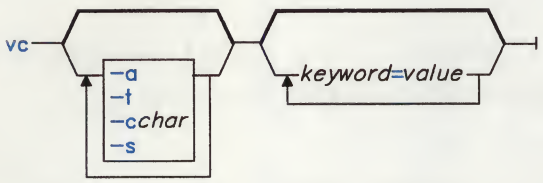
The **system** and **filesystems** files in *AIX Operating System Technical Reference.*

vc

Purpose

Substitutes assigned values in place of keywords.

Syntax



OL805293

Description

The **vc** command is used to control writing different versions of a file to standard output. However, since Source Code Control System commands ("**admin**" on page 41, "**get**" on page 477, and "**delta**" on page 310) provide more efficient and complete control, they should be used instead of **vc**.

The **vc** command copies lines from standard input to standard output. The flags and *keywords* on the command line and control statements in the input modify the resulting output. **vc** replaces user declared *keywords* with their *value* assigned on the command line. These *keywords* can be replaced both in text and in control statements.

Control Statements

A control statement is a single line beginning with a control character (the default control character is a : (colon)). They provide conditional processing of the input. The allowable types of control statements are:

:if *condition*
text
:end

Writes all the lines between the **:if** statement and the matching **:end** to standard output only if *condition* is true. You can nest **:if-:end** statements, but once a condition is false, all remaining nested **:if-:end** statements are ignored. See "Condition Syntax" on page 1183 for the syntax of conditions and allowable operators.

:dcl *keyword* [, *keyword* . . .]
Declares *keyword*s. All *keyword*s must be declared.

:asg *keyword*=*value*
Assigns *value* to *keyword*. An **:asg** statement takes precedence over keyword assignment on the **vc** command line. A later **:asg** statement overrides all earlier assignments of the associated *keyword*. *keyword*s declared, but not assigned *values* have null values.

:: *text*
Removes the two leading control characters and replaces *keyword*s with their *value*s, and then copies the line to the standard output.

:on
:off
Turns on or off *keyword* replacement on all lines.

:ctl *char*
Changes the control character to *char*.

:msg *message*
Writes a message to standard error output in the form:

`Message(`*n*`):` *message*

where *n* is number of the input line on which the message appeared.

:err *message*
Writes an error message to standard error in the form:

```
ERROR: message
ERROR: err statement on line n (vc15)
```

vc stops processing and returns an exit *value* of 1.

Condition Syntax

Item	Statements Allowed
condition	::=or statement
	::=not or statement
or statement	::=and statement
	::=and statement ¦ or statement
and statement	::=expression
	::=expression & and statement
expression	::=(or statement)
	::=value operator value
operator	::== or != or < or >
value	::=ASCII string

Item	Statements Allowed
	: :=numeric string

The available condition operators and their meanings are as follows:

=	equal
!=	not equal
&	and
¦	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition.

The > and < (greater than and less than symbols) operate only on unsigned integer values; for example: 012 > 12 is false. All other operators take strings as modifiers; for example 012 != 12 is true. The precedence of the operators, from highest to lowest precedence, is as follows:

= != > < all of equal precedence

&

¦

Parentheses can be used, of course, to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

Keyword Replacement

A *keyword* must begin and end with the same control character used in control statements. A *keyword* may be up to nine alphanumeric characters, where the first character must be alphabetic. Keyword *values* can be any ASCII string. A numeric keyword *value* is an unsigned string of digits. *values* cannot contain tabs or spaces.

Examples of *keyword*=*value* assignments are:

```
numlines=4
prog=acctg
pass4=yes
```

The **vc** command removes all control characters and *keyword*s from input text lines marked with two control characters as it writes the text to standard output. To prevent a control character from being interpreted, precede it with a backslash, as in the following example:

```
::the :prog: program includes several of the following\:
```

The keyword **:prog:** is replaced by its *value*, but the **\:** is passed to standard output as **:** (colon).

Input lines beginning with a \ (backslash) followed by a control character are not control lines, and are copied to standard output without the backslash. **vc** writes lines beginning with a backslash and no following control character without any changes (including the initial backslash).

Flags

-a Replaces *keyword*s surrounded by control characters with their assigned *value* in all text lines (not just those beginning with two control characters).

-c*char* Uses *char* as the control character.

Japanese Language Support Information

char must be an ASCII character.

-s Does not display the warning messages normally displayed to standard error.

-t Ignores all characters from the beginning of a line up to and including the first tab character for detecting a control statement. If **vc** finds a control character, it ignores all characters up to and including the tab.

Related Information

The following commands: "**admin**" on page 41, "**delta**" on page 310, and "**get**" on page 477.

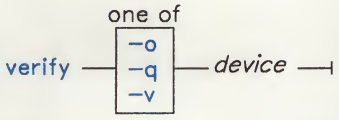
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

verify

Purpose

Turns write verification on or off for a particular minidisk.

Syntax



OL805446

Description

The **verify** command turns write verification on or off for the specified minidisk *device*. The name must be a stanza name in the **/etc/system** file.

When write verification is on, the system checks each write operation to the minidisk by comparing the data written to disk with the data in the write buffer. If it detects an uncorrectable error, then it passes an error code back from the write operation.

At system startup, write verification is turned off.

Flags

-o Turns write verification off.

-q Tells you whether write verification is set on or off.

-v Turns write verification on.

Files

/dev/config
/etc/system

Related Information

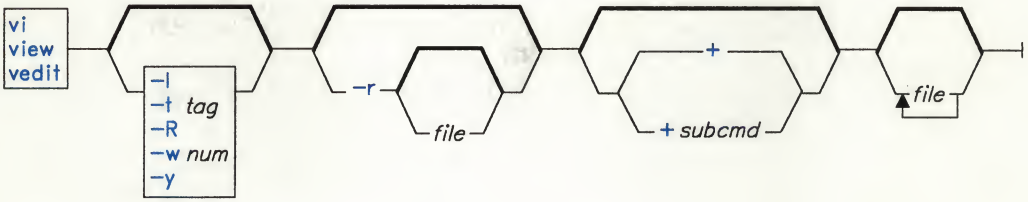
The **mdverify** subroutine in *AIX Operating System Technical Reference*.

vi, vedit, view

Purpose

Edits files with a full screen display.

Syntax



OL805424

Description

The **vi** command starts a display editor based on an underlying line editor (**ex**). The **view** command is a read-only version of **vi**. In it, the **readonly** option is set to protect files during browsing. The **vedit** command is a version of **vi** intended for beginners. In it, the **report** option is set to 1, and the **showmode** and **novice** options are set. Since **novice** is set, it is a line editor. For more information on these options, see "Setting Options" on page 1189.

The *file* parameter specifies the file or files to be edited. If you supply more than one *file* on the command line, **vi** edits each file in the specified order.

When you use **vi**, changes made to a file are reflected automatically in your display. The position of the cursor on the display indicates its position within the file. The subcommands affect the file at the cursor position.

The following list provides the maximum limits of the **vi** editor. These counts assume single-byte characters.

- 2048 characters per line
- 256 characters per global command list
- 128 characters in the previous inserted and deleted text
- 128 characters in a shell escape command
- 128 characters in a string-valued option
- 30 characters in a tag name
- 1,048,560 lines of 2048 characters per line silently enforced
- 128 map macros with 2048 characters total.

Editing States

The **vi** editor has the following operational states:

command state
: This is the initial state. Any subcommand can be entered (except commands that can only be used in the text input state). When subcommands and other states end, they return to this state. Pressing **Esc** cancels a partial command.

text input state
: You use **vi** in this state when you add text. Entered this state with the **a**, **A**, **i**, **I**, **o**, **O**, **c**, **C**, **s**, **S**, and **R** subcommands. After entering one of these commands, you can enter text into the editing buffer at the current cursor position. To return to the command state, press **Esc** for normal exit or press **INTERRUPT** (**Alt-Pause**) to end abnormally.

last line state
: Some subcommands (subcommands with the prefix **:**, **/**, **?**, **!***obj*, or **!!**) read input on a line displayed at the bottom of the screen. When you enter the initial character, **vi** places the cursor at the bottom of the screen, where you enter the remaining characters of the command. Press the **Enter** key to run the subcommand and **INTERRUPT** (**Alt-Pause**) to cancel it. When **! !** is used, the cursor moves only after both **!**s are entered. When you use **:** (colon) to enter the last line state, special meaning is given to the following characters when they are used before commands which specify counts.

%	All lines regardless of cursor position
$	Last line
.	Current line

Character Sets with vi

The collation table defines the alphanumeric set used by your system. This table affects the performance of **vi** macros and subcommands. If you intend to use non-ASCII extended characters with **vi** macros, it may be necessary to revise this table using the **ctab** command.

The **vi** editor uses the collation table to distinguish between a ***small word*** and a ***big word***. A small word is bounded by alphabetic characters or numbers that are not defined in the collation table. For example, `isn't` is two small words. The **'** (apostrophe) is not a number or an alphabetic character, and it bounds both the small word `t` and the small word `isn`. A big word is bounded by blanks, tabs, and new-line indicators. For example, `stop` is a big word. For more information see "**ctab**" on page 257 in this book. For more information on extended characters, international characters, and collation tables see *Managing the AIX Operating System*.

Setting Options

The **vi** editor allows you to customize options so that you can use the editor for a specific task. Use the **set** command to set or change an option. You set some options to a string or a number value, other options you simply turn on or off. To change an option set to a value, enter a command in the form **:set** *option*=*value*. To toggle an option that can be set to on or off, enter a line of the form **:set** *option* to set it on or **:set no** *option* to set it off. To view the current setting of all the options, enter `:set all` while in the **vi** command state.

You can abbreviate options with the **set** command. The following table lists commonly used options, abbreviations, and descriptions:

Option	Abbreviation	Description
autoindent	**ai**	Indents automatically in text input mode to the indentation on the previous line by using the spacing between tab stops specified by the **shiftwidth** option. The default is **noai**. To back the cursor up to the previous tab stop, type **Ctrl-D**. This option is not in effect for global commands.
autoprint	**ap**	Prints the current line after any command that changes the editing buffer. The default is **ap**. This option applies only to the last command in a sequence of commands on a single line, and is not in effect for global commands.
autowrite	**aw**	Writes the editing buffer to the file automatically before the **:n**, **:ta**, **Ctrl-A**, and **!** subcommands if the editing buffer changed since the last **write** command. The default is **noaw**.
beautifying text	**bf**	Prevents the user from entering control characters (except for tab, new-line, and scans form feed) in the editing buffer during text entry. The default is **nobf**. This option applies to command input.
closepunct	**cp=**	Handles a list of closing punctuation specially when wrapping text (see **wraptype**). Precedes multi-character punctuation with the number of characters. Example: `cp=3.;)}`. The **vi** command does not split closing punctuation when wrapping. To set the default value, run kanji-compatible **vi** and enter **:set cp**.
directory	**dir=**	Displays the directory that contains the editing buffer. The default is **dir=/tmp**.

Option	Abbreviation	Description
edcompatible	**ed**	Retains global (**g**) and confirm (**c**) subcommand suffixes during multiple substitutions and causes the read (**r**) suffix to work like the **r** subcommand. The default is **noed**.
hardtabs	**ht=**	Tells **vi** the distance between the hardware tab stops on your display. The default is **ht=8**.
ignorecase	**ic**	Ignores distinction between upper- and lower-case while searching for regular expressions. The default is **noic**.
lisp	**lisp**	Removes the special meaning of (), {}, [[and]] and enables the = (formatted print) operator for S-expressions, so you can edit LISP programs. The default is **nolisp**.
list	**list**	Displays text with tabs and the end of lines marked. Tabs display as ^I and the end of lines as $. The default is **nolist**.
magic	**magic**	Treats the characters . (period), [, and * as special characters when scanning. In off mode, only the ( ) and $ retain special meanings. However, you can evoke special meaning in other characters by preceding them with a \ (back slash). The default is **magic**.
modeline	**modeline**	Runs an editor command line if found in the first five or the last five lines of the file. An editor command line can be anywhere in a line. For the editor to recognize a command line, the line must contain a space or a tab followed by the string `ex:` or `vi:`. The command line is ended by a second : (colon). The editor tries to interpret any data between the first and second colon as editor commands. The default is **nomodeline**.
number	**nu**	Displays lines prefixed with their line numbers. The default is **nonu**.
optimize	**opt**	Speeds up the operation of terminals that lack cursor-addressing. The default is **noopt**.
paragraphs	**para=**	Defines **vi** macro names that start paragraphs. The default is **para=IPLPPPQPP LIpplpipnpbp**. Single-letter **nroff** macros, such as **.P**, must include the space as a quoted character if respecifying a paragraph.

Option	Abbreviation	Description
partialchar	**pc=**	Appears in the last display column where a double-wide character would not display completely. The default character is – (dash).
redraw	**redraw**	Simulates a smart work station on a dumb work station. The default is **nore**.
report	**report=**	Sets the number of times you can repeat a command before a message is displayed. For subcommands that produce many messages, such as global subcommands, the messages are displayed when the command sequence completes. The default is **report=5**.
scroll	**scr=**	Sets the number of lines to be scrolled when the user scrolls up or down. The default is **scroll=12**.
sections	**sect=**	Defines to **vi** macro names that start sections. The default is **sect=NHSHH HUuhsh+c**. Single-letter **nroff** macros, such as **.P**, must include the space as a quoted character if respecifying a paragraph.
shell	**sh=**	Defines the shell for **!** or **:!** commands. The default is the login shell.
shiftwidth	**sw=**	Sets the distance for the software tab stops used by **autoindent**, the shift commands (> and <), and the text input commands (**Ctrl-D** and **Ctrl-T**). The default is **sw=8**.
showmatch	**sm**	Shows the matching open parenthesis (or open bracket { as you type the close parenthesis) or close bracket }. The default is **nosm**.
slowopen	**slow**	Postpones updating the display during inserts. The default is **noslow**.
tabstops	**ts=**	Sets distance between tab stops in a displayed file. The default is **ts=8**.
term	**term=**	Sets the type of work station you are using. The default is **term=$TERM** where **$TERM** is the value of the shell variable **TERM**.

Option	Abbreviation	Description
terse	**terse**	Allows **vi** to display the short form of messages. The default is **noterse**.
timeout	**to**	Sets a time limit of two seconds on entry of characters. This limit allows the characters in a macro to be entered and processed as separate characters when **timeout** is set. To resume use of the macro, set **notimeout**. The default is **to**.
warn	**warn**	Displays a warning message before the **!** subcommand executes a shell command if it is the first time you issued a shell command after changes were made in the editing buffer but not written to a file. The default is **warn**.
window	**wi=**	Sets the number of lines displayed in one window of text. The default is dependent on the baud rate at which you are operating: 600 baud or less / 8 lines, 1200 baud / 16 lines, higher speeds / full screen minus one.
wrapmargin	**wm=**	Sets the margin for automatic word wrapping from one line to the next. The default is **wm=0**. A value of zero turns off word wrapping.
wrapscan	**ws**	Allows string searches to wrap from the end of the editing buffer to the beginning. The default is **ws**.
wraptype	**wt=**	Determines how a line splits for word wrapping. The default is **general**. The default causes **vi** to insert a new line after a white space. Any characters following the insert point begin the new line. White space immediately before the inserted line is deleted. When Japanese Language Support is installed and **wraptype=general**, **vi** inserts a new line to wrap one character. One character forming closing punctuation is allowed past the right margin.
writeany	**wa**	Turns off the checks usually made before a **write** command. The default is **nowa**.

Defining Macros

If you use a subcommand or sequence of subcommands frequently, you can create a macro that will issue the subcommand or sequence. To create a macro, enter the sequence of subcommands into an editing buffer named with a letter of the alphabet. When used, **a - z** overlay the contents of the buffer; **A - Z** append text to the previous contents of the buffer, allowing the building of a macro piece by piece.

To invoke the macro, enter `@`*x* where *x* is the letter name of the buffer. Enter `@@` to repeat the last macro you invoked.

Mapping Keys

You can use the **map** command to map a keystroke to a subcommand or a sequence of subcommands. To map a key, enter **:map** *key subcommand* where *key* is the key you want to assign to a subcommand or sequence of subcommands, and *subcommand* is the subcommand or sequence of subcommands. For example, to map `@` to delete lines, enter:

```
:map  @  dd
```

In this example, `@` is the key to which the subcommand is assigned and `dd` is the subcommand.

In the next example, a subcommand sequence is mapped to a key:

```
:map * {>}
```

The * is the key to which the subcommand sequence is assigned and {>} is the subcommand sequence. The { moves the cursor to the beginning of the paragraph and the > indents the paragraph to the next shiftwidth.

While in text input state, enter the command **:map!** to display a list of the current key mappings. To remove a key map, enter **:unmap** *string* or **:unmap!** *string*, where *string* is the string used after the **:map** command to set the key and subcommand sequence. For example, to remove key map for the previous example, enter:

```
:unmap * {>}
```

If function keys are defined for your terminal, you can include them in the **map** or **unmap** command by typing **Ctrl-V** before pressing the desired key. It is useful to define keys seldom used in editing with another key or function key (**F0 - F12**). For example, the **Shift**, **Ctrl**, or **Alt** can be defined in this way.

The abbreviation command **:ab** is similar to the **:map** command. For example, if you set the letter `s` equal to `Sam` with a **:map** command and then entered the following sentence,

```
s ate apples.
```

it would display as

```
Sam ate applesam.
```

The **:map** command does not recognize the difference between the macro `s` and the text `s.` With some restrictions, the **:ab** command does distinquish between text and a macro. Setting the macro in the previous example with the **:ab** command,

```
:ab s Sam
```

and typing the same sentence would result in the correct sentence, `Sam ate apples.` The **:ab** command only recognizes a macro when it is proceeded by a blank space or tab character. If the following were entered,

```
Sam swims
```

the **ab** macro would translate the sentence as:

```
Sam Samwims
```

The abbreviated item can occupy more than one line. However, you must use the **ex** editor to remove a macro that contains a **Ctrl-M** (enter sequence). After entering **ex**, issue a **:unab** *abbreviation*, and return to **vi**. Remove macros without the **Ctrl-M** by using **:ab** *abbreviation*. After removing or changing abbreviations created with **:ab**, enter **:ab** to list all currently defined abbreviations.

If you use an IBM 3161 ASCII Display Station, IBM 3163 ASCII Display or IBM 3101 ASCII Display Station, the default key mapping of **vi** can cause you to lose data. To see the default mapping, issue a **:map** command. Specific problems arise with the **Esc-J** or **Shift-J** sequence. This sequence deletes any information from the current position of the cursor to the end of the file. To avoid problems, change this sequence using a **.exrc** file.

Keeping a Customized Change

When you customize **vi** from the **vi** command line, the customization is in effect until you leave the editor. If you want to keep your assignments you must put the commands in the file **.exrc**. Each time you start the editor it reads this file. When you create the customization file, do not type the : (colon) before each command. The : is only required when entering commands on the command line. The following is an example of a **.exrc** file:

```
set ai aw
set wm=5
map  @  dd
```

Flags

-l	Enters **vi** in LISP mode. In this mode, **vi** indents appropriately for LISP code and the **(**, **)**, {, }, **[[**, and **]]** subcommands are modified to act appropriately for LISP.
-r [*file*]	Recovers a file after an editor or system crash. If you do not specify *file*, **vi** displays a list of all saved files.

-R	Sets the **readonly** option to protect the *file* against overwriting.
-t *tag*	Edits the file containing the *tag* and positions the editor at its definition.
-w*num*	Sets the default window size to *num*. This is useful when you use the editor over a low-speed line.
+[*subcmd*]	Carries out the **ex** subcommand before editing begins. If you do not specify *subcmd*, the cursor is placed on the last line of the file.
-y	Overrides the maximum line setting of 1,048,560 with any value above 1024.

Subcommands

In the following lists, **Esc** stands for pressing the **ESCAPE** key instead of pressing the **Enter** key.

General Subcommand Syntax

[*named_buffer*] [*operator*] [*number*] *object*

Surrounding square brackets indicate optional items.

[*named_buffer*]	A temporary text storage area in memory.
[*operator*]	Specifies the subcommand or action; instructs **vi**.
[*number*]	A whole decimal value that specifies either the extent of the action, or a line address.
object	Specifies what to act on. This can be a text object (a character, word, sentence, paragraph, section, character string) or a text position (a line, position in the current line, screen position).

Counts before Subcommands

You can prefix many subcommands with a number. The **vi** editor interprets this number in one of the following ways:

1. Go to line *number*:

```
5G
10z
```

2. Go to column *number*:

```
25|
```

3. Scroll *number* lines:

```
10Ctrl-D
10Ctrl-U
```

Subcommands for Moving within the File

There are many commands that you can use to move within a file. They can be entered while **vi** is in the command state.

Movements within a Line

← or **h** or **Ctrl-H**
: Moves the cursor one character to the left.

↓ or **j** or **Ctrl-J** or **Ctrl-N**
: Moves the cursor down one line (but it remains in the same column).

↑ or **k** or **Ctrl-P**
: Moves the cursor up one line (but it remains in the same column).

→ or **l**
: Moves the cursor one character to the right.

Character Positioning within a Line

^	Moves the cursor to the first nonblank character.
0	Moves the cursor to the beginning of the line.
$	Moves the cursor to the end of the line.
f*x*	Moves the cursor to the next *x* character.
F*x*	Moves the cursor to the last *x* character.
t*x*	Moves the cursor to one column before the next *x* character.
T*x*	Moves the cursor to one column after the last *x* character.
;	Repeat the last **f**, **F**, **t**, or **T** subcommand.
,	Repeat the last **f**, **F**, **t**, or **T** subcommand in the opposition direction.
*num***|**	Moves the cursor to the specified column.

Words, Sentences, Paragraphs

w	Moves the cursor to the next small word.
b	Moves the cursor to the previous small word.
e	Moves the cursor to the end of the small word
W	Moves the cursor to the next big word.

B Moves the cursor to the previous big word.

E Moves the cursor to the end of a big word.

Line Positioning

H Moves the cursor to the top line on the screen.

L Moves the cursor to the last line on the screen.

M Moves the cursor to the middle line on the screen.

+ Moves the cursor to the next line at its first nonblank character.

- Moves the cursor to the previous line at its first nonblank character.

Enter Moves the cursor to the next line at its first nonblank character.

Scrolling

Ctrl-U Scrolls up one half screen.

Ctrl-D Scrolls down one half screen.

Ctrl-F Scrolls forward one screen.

Ctrl-B Scrolls backward one screen.

Searching for Patterns

[*num*]**G** Places the cursor at line number *num* or to the last line if *num* is not specified.

/pattern Places the cursor at the next line containing *pattern*.

?*pattern* Places the cursor at the next previous line containing *pattern*.

n Repeats last search for *pattern* in the same direction.

N Repeats last search for *pattern* in the opposite direction.

/pattern/+*num*
Places the cursor at the *num*th line after the line matching *pattern*.

?*pattern***?**-*num*
Places the cursor at the *num*th line before the line matching *pattern*.

% Finds the parentheses or brace that matches the one at the current cursor position.

Moving to Sentences, Paragraphs, or Sections

]]	Places the cursor at next section (or function if you are in LISP mode).
[[	Places the cursor at previous section (or function if you are in LISP mode).
(	Places the cursor at the beginning of the previous sentence (or the previous s-expression if you are in LISP mode).
)	Places the cursor at the beginning of the next sentence (or the next s-expression if you are in LISP mode).
{	Places the cursor at the beginning of the next paragraph (or at the next list if you are in LISP mode).
}	Places the cursor at the beginning of the next paragraph, at the next section if you are in C mode, or at the next list if you are in LISP mode.

Marking and Returning

``	Moves the cursor to the previous location of the current line.
''	Moves cursor to the beginning of the line containing the pervious location off the current line.
m*x*	Marks the current position with letter *x*.
`*x*	Moves cursor to mark *x*.
'*x*	Moves cursor to the beginning of the line containing mark *x*.

Adjusting the Screen

Ctrl-L	Clears and redraws the screen.
Ctrl-R	Redraws the screen and eliminates blank lines marked with a @.
z	Redraws the screen with the current line at the top of the screen.
z-	Redraws the screen with the current line at the bottom of the screen.
z.	Redraws the screen with the current line at the center of the screen.
/*pattern*/**z-**	Redraws the screen with the line containing *pattern* at the bottom.
z*num*	Makes the window *num* lines long.
z+	Page Up.
z^	Page Down.

Ctrl-E Scrolls the window down one line.

Ctrl-Y Scrolls the window up one line.

Subcommands for Editing

Use the following subcommands to edit your text. Subcommands not followed by an * (asterisk) should be entered in the text input state. These subcommands affect the text relative to the current cursor position. You return to the command state by pressing the **Esc** key.

Editing the File

a*text* Inserts *text* after the cursor.

A*text* Adds *text* to the end of the line.

C Changes rest of line (**c$**).

cc Changes a line.

cw Changes a word.

cw*text* Changes word to *text*.

D * Deletes the rest of the line (**d$**).

dd * Deletes a line.

dw * Deletes a word.

i*text* Inserts *text* before the cursor.

I*text* Inserts *text* before the first nonblank character in the line.

J * Joins lines.

o Adds an empty line below the current line.

O Adds an empty line above the current line.

r*x* * Replaces the current character with *x*. (Commands followed by * do not enter the text input state.)

R*text* Overwrites characters with *text*.

s * Substitutes characters (**cl**).

S * Substitutes lines (**cc**).

u * Undoes the previous change.

x * Deletes a character.

X *	Deletes characters before cursor (**dh**).
yw *	Yanks a word into the **undo** buffer.
yy *	Yanks a line into the **undo** buffer.
< < *	Shifts one line to the left.
<L *	Shifts all lines from the cursor to the end of the screen to the left.
> > *	Shifts one line to the right.
>L *	Shifts all lines from the cursor to the end of the screen to the right.
~ *	Changes letters at cursor to opposite case.
! *	Indents for LISP.

Corrections during Insert

Use the following commands only while in text input state. They have different meanings in the command state.

Ctrl-H	Erases last character.
Ctrl-W	Erases last small word. **Note:** For more information on small words see "Character Sets with **vi**" in this section.
****	Quotes the erase and kill characters.
Esc	Ends insertion, back to command state.
Ctrl-?	Interrupts, terminates insert or **Ctrl-D**.
Ctrl-D	Goes back to previous **autoindent** stop.
^Ctrl-D	Ends **autoindent** for this line only.
0Ctrl-D	Moves cursor back to left margin.
Ctrl-V	Enters nonprinting character.

Moving Text

p	Puts back text in the **undo** buffer after the cursor.
P	Puts back text in the **undo** buffer before the cursor.
"*x*p	Puts back text from the buffer *x*.

"*x*d	Deletes text into the buffer *x*.
y	Places the object that follows (for example, **w** for word) in the **undo** buffer.
"*x*y	Places the object that follows in the *x* buffer, where *x* is any letter.
Y	Places the line in the **undo** buffer.

Restoring and Repeating Changes

u	Undoes the last change.
U	Restores the current line if the cursor has not left the line since the last change.
.	Repeats the last change or increments the **"*n* p** command. **Note:** This subcommand is not meant for use with a macro. Enter **@@** to repeat a macro.
"*n* p	Retrieves the *n*th last delete of a complete line or block of lines.

Interrupting, Canceling, and Exiting vi

Q	Enter **ex** editor in command state.
:sh	Runs a shell. You can return to **vi** by pressing **Ctrl-D**.
:!*cmd*	Runs *cmd*, then returns.
:!!	Repeats the last **:!*cmd*** command.
n*!!*cmd	Executes shell command *cmd* and replaces the number of lines specified by *n* with the output of *cmd*. If *n* is not specified, when the default is 1. If *cmd* expects standard input, the lines specified are used as input. Thus the command **!sort** can sort a paragraph.
n*!*obj cmd	Executes shell command *cmd* and replaces *n* with output of *cmd*. If *n* is not specified, the default is 1. If *cmd* expects standard input, the lines or *obj* specified is used as input.
ZZ	Exits **vi**, saving changes.
:q	Quits **vi**. If you have changed the contents of the editing buffer, **vi** displays a warning message and does not quit.
:q!	Quits **vi**, discarding the editing buffer.
Esc	Ends insert or ends an incomplete subcommand.
Ctrl-L	Redisplays a screen.
Ctrl-R	Redisplays the screen if **Ctrl-L** is the → key.

Ctrl-? Interrupts a subcommand.

File Manipulation

:e *file* Edits *file*.

:e! Re-edits the current file and discards all changes.

:e + *file* Edits *file* starting at the end.

:e +*num* Edits *file* starting at line *num*.

:e # Edits the alternate file. The alternate file is usually the previous current file name. However, if changes are pending on the current file when a new file is called, the new file becomes the alternate file. This subcommand is the same as **Ctrl-A**.

:n Edits next file in the list entered on the command line.

:n *files* Specifies new list of files to edit.

:r *file* Reads the file into the editing buffer by adding new lines below the current cursor position.

:r !*cmd* Runs the shell command *cmd* and places its output in the file by adding new lines below the current cursor position.

:ta *tag* Edits a file containing *tag* at the location of *tag*.

:w Writes the editing buffer contents to the original file.

:w *file* Writes the editing buffer contents to the named *file*.

:w! *file* Overwrites *file* with the editing buffer contents.

Ctrl-G Shows current file name and line.

Ctrl-A. Edits the alternate file. The alternate file is usually the previous current file name. However, if changes are pending on the current file when a new file is called, the new file becomes the alternate file. This subcommand is the same as **:e #**.

Related Information

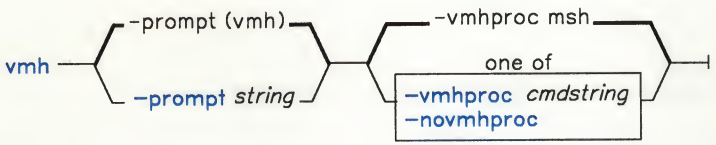
The following commands: "**ed**" on page 371 and "**ex**" on page 407.

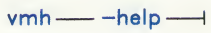
vmh

Purpose

Invokes a visual interface for use with MH commands.

Syntax





AJ2FL239

Description

The **vmh** command is used to invoke a visual interface for use with MH commands. **vmh** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **vmh** command implements the server side of the MH window management protocol and maintains a split-screen interface to any program that implements the client side of the protocol.

vmh prompts for commands and sends them to the client side of the protocol. If the command produces a window of more than one screen's worth of output, **vmh** prompts the user for a subcommand.

Subcommands

SPACE	Advances to the next screen.
[*num*] **ENTER**	Advances the specified number of lines. The default is one line.
[*num*] **y**	Goes back the specified number of lines. The default is one line.
[*num*] **d**	Advances ten times the specified number of lines. The default for *num* is 1, for a total of ten lines.

[*num*] **u**	Goes back ten times the specified number of lines. The default for *num* is 1, for a total of ten lines.
[*num*] **g**	Goes to the specified line.
[*num*] **G**	Goes to the end of the window. If *num* is specified, this command acts like **g**.
CTRL-L	Refreshes the screen.
h	Displays a help message.
q	Ends output.

Flags

-help	Displays help information for the command.
-novmhproc	Runs the default *vmproc* directly, without the window management protocol.
-prompt *string*	Uses the specified string as the prompt.
-vmhproc *cmdstring*	Specifies the program which implements the client side of the window management protocol. The default is **msh**.

Profile Entries

Path:	Specifies your **mhpath**.
mshproc:	Specifies the program used for the MH shell.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

The MH command "**msh**" on page 677.

The **mh-profile** file in *AIX Operating System Technical Reference.*

The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

vrm2rtfont

Purpose

Converts a standard AIX font file to RT rtx font format.

Syntax

AJ2FL125

Description

Before any of the standard fonts provided with the AIX Operating System can be used with the X-Windows program, they must be converted to the X-Windows' **rtx** font file-format. This command takes an AIX font file as input and converts the file to **rtx** format. This format can be used with X-Windows on RT displays.

Examples

To convert the AIX font file **etc/vtm/itl1.9x20** to an **rtx** font file:

```
vrm2rtfont /etc/vtm/itl1.9x20  /usr/lpp/fonts/Itl14.500
```

Note how the output file name conforms to the **rtx** naming convention.

Files

/etc/vtm	Contains AIX fonts.
/usr/lpp/fonts	Contains fonts for other programs.

Related Information

The following command: "**vrm2rtfont**."

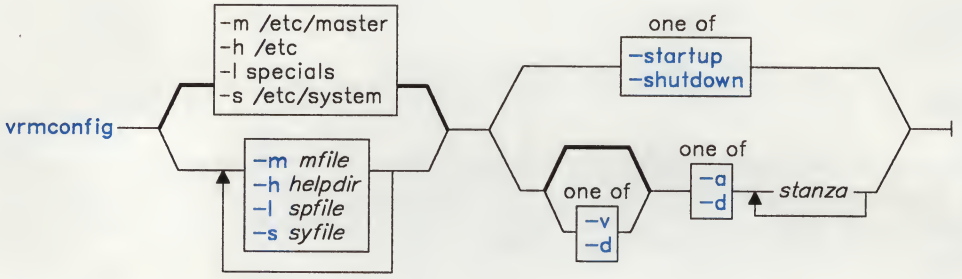
For more information on X-Windows, see *X-Windows*.

vrmconfig

Purpose

Installs peripheral devices.

Syntax



OL805400

Description

The **vrmconfig** command installs VRM device drivers. Normally, it runs automatically at each system startup. Its exit value is the number of errors it encountered. When auditing is on, an audit record of the type, **vrmconfig** is created.

Note: Most users will never need to run this command from the command line.

The **vrmconfig** command performs its operations through the **/dev/config** driver.

Flags

-a *stanza* Adds devices to a running system. **vrmconfig** processes the specified *stanza* in **/etc/system** or the file specified with the **-s** flag.

-d *stanza* Deletes a device from a running system. **vrmconfig** processes the specified *stanza* in **/etc/system** or the file specified with **-s**. The special file list produced includes commands to remove relevant special files, since **vrmconfig** assumes that the device has been removed from the configuration.

-h *helpdir* Specifies the directory that contains the configuration helper programs. The default value is **/etc**.

-l *spfile* Specifies the output special file list. The default value is **specials**.

-m *mfile* Specifies the input master configuration file. The default value is **/etc/master**.

-s *syfile* Specifies the input system configuration file. The default value is **/etc/system**.

-u Causes only kernel-related configuration steps to be performed, that is, kernel driver installation calls. This flag may be used only with the **-a** or **-d** flags.

-v Causes only VRM-related configuration steps to be performed, that is, **definedevice** calls. This flag may be used only with the **-a** or **-d** flags.

-shutdown Causes all installed drives to be deleted for shutting down the system. Special files are not deleted, since the actual configuration is not considered changed.

-startup Causes all defined devices to be configured in at system startup. (Any stanza that contains the attribute **noipl=true** is skipped.) For devices such as minidisks, which remain configured between system restarts, **vrmconfig** does not perform "once-only" configuration steps. It does not modify special files that already exist.

Files

/etc/confgstatus	Status file recording current configuration status.
/etc/system	Default system file.
/etc/master	Default master file.
specials	Default special file list.
/etc	Default directory containing configuration helper programs.
/etc/vrmdd	Directory containing VRM device driver modules.
/vrm/vrmdd	Directory containing VRM device driver modules.
/etc/ddi	Directory containing device specific information files.

Related Information

The following command: "**config**" on page 194.

The **master** and **system** files in *AIX Operating System Technical Reference.*

Installing and Customizing the AIX Operating System.

wall

Purpose

Writes a message to all logged-in users.

Syntax

OL805017

Description

The **wall** command reads a message from standard input until it reaches an end-of-file character. It then sends the message to all logged-in users preceded by the following heading:

```
Broadcast Message from user
```

To override any protections other users have set up, you must be operating with superuser authority. Typically, **the superuser** uses **wall** to warn all users of an impending system shutdown.

Note: This command only sends messages to the local node.

Files

/dev/tty*

Related Information

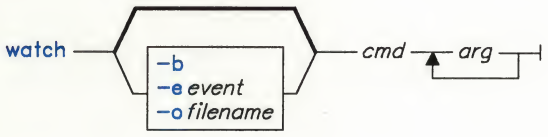
The following commands: "**mesg**" on page 642, "**su**" on page 1026, and "**write**" on page 1225.

watch

Purpose

Observes and reports security-relevant actions on the system.

Syntax



OL805487

Description

The **watch** command allows a user with superuser authority to observe the actions of an untrustworthy program.

The **watch** command creates a channel to the audit device on which the events specified by **-e** *event* are recorded. If no *event*s are specified, a default set of events is audited and appears on this device. The **watch** command reads these audit records and writes them to standard output.

The **watch** command executes the command specified by *cmd*, (with the arguments specified by *arg*s). If the *cmd* or any of its descendants perform an action specified by the *event*, the action is audited.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-b Specifies that audit records should be written out as binary records. Normally, the **watch** command first processes audit records through **auditpr**.

-e *event* Specifies an event to be audited for the command. If no **-e** flag is specified, a list of events which contains all modifications of the trusted computing base is audited.

-o *filename* Specifies a file name for output. If the **-o** flag is not specified, audit records are written to standard output.

Files

/dev/audit Used to monitor and enable the actions of descendent processes.

Related Information

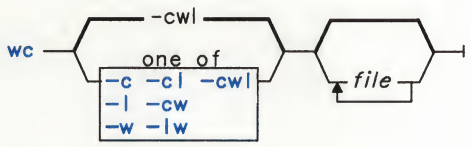
The discussion of trusted programs and the trusted computing base in *Managing the AIX Operating System.*

wc

Purpose

Counts the number of lines, words, and characters in a file.

Syntax



OL805242

Description

The **wc** command counts the number of lines, words, or characters in *file* or in the standard input if you do not specify any *file*s. It writes the results to standard output. It also keeps a total count for all named files. A word is defined as a string of characters delimited by spaces, tabs, or new-line characters. **wc** counts lines, words, and characters by default.

When you specify more than one *file* on the command line, **wc** displays the name of the file along with the counts.

Flags

-c Counts bytes.

-l Counts lines.

-w Counts words.

Examples

1. To display the line, word, and character counts of a file:

   ```
   wc chap1
   ```

   This displays the number of lines, words, and characters in the file `chap1`.

2. To display only character and word counts:

```
wc -cw chap*
```

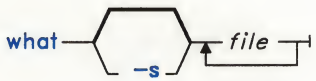
This displays the number of characters and words in each file whose name starts with `chap`, and displays the totals.

what

Purpose

Displays identifying information in files.

Syntax



OL805295

Description

The **what** command searches the named *file*s for all occurrences of the pattern that **get** substitutes for the %**Z**% keyletter (see "Identification Keywords" on page 480). By convention, the value substituted is `@(#)`.

what writes to standard output whatever follows the pattern up to but not including the first double quotation mark ("), greater than symbol (>), new-line character , backslash (\), or null character.

The **what** command is intended for use in conjunction with the **get** command, which automatically inserts the identifying information. You can also use **what** on files where the information is inserted manually.

Flags

-s Searches for only the first occurrence of `@(#)`.

Examples

Suppose that the file `test.c` contains a C program that includes the line:

```
char ident[ ] = "@(#)Test Program";
```

If you compile `test.c` to produce `test.o` and `a.out`, then the command:

```
what test.c test.o a.out
```

displays:

```
test.c:
        Test Program
test.o:
        Test Program
a.out:
        Test Program
```

Related Information

The following commands: "**get**" on page 477, and "**help**" on page 513.

The **sccsfile** file in *AIX Operating System Technical Reference.*

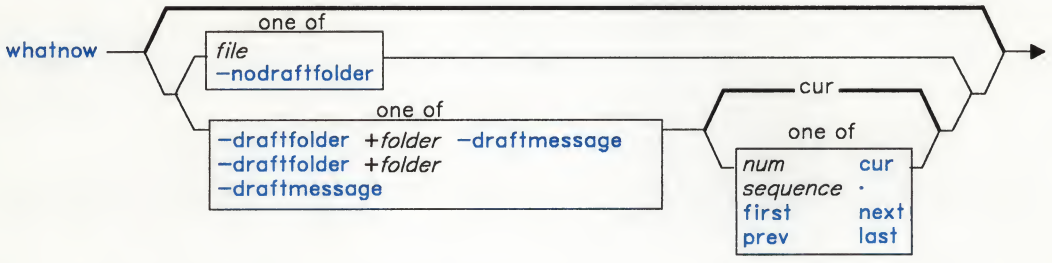
The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces.*

whatnow

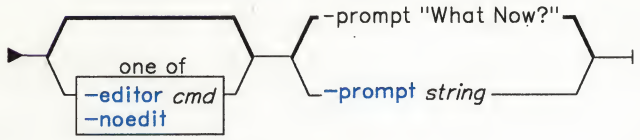
Purpose

Invokes a prompting interface for draft disposition.

Syntax



AJ2FL244



AJ2FL158

Description

The **whatnow** command is the default program used by **comp**, **dist**, **forw**, and **repl** to prompt you for the disposition of messages. **whatnow** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

By default, **whatnow** invokes an editor and places the current draft message (or *file*) in the editing session. When you exit the editing session, **whatnow** prompts: `What now?` You can specify any of the **whatnow** subcommands, or you can press **Enter** to see a list of the subcommands. These subcommands enable you to re-edit the message, direct the disposition of the message, or end the processing of the **whatnow** command. The **whatnow** command continues to prompt you for subcommands until you specify **quit**.

Subcommands

display [*flags*]	Displays the message being acted upon (redistributed or replied to). For *flags*, you can specify any flag that is valid for the command serving as the *lproc*. (You can set a default **lproc:** entry in **$HOME/.mh_profile**.) If you specify any flags that are not valid for *lproc*, **whatnow** does not pass the path name of the draft to the **lproc**.
edit [*cmdstring*]	Re-edits the message using the specified editor. You can specify the editor and any valid flags to that editor. If you do not specify an editor, **whatnow** selects a default editor according to the information supplied in the MH profiles. You can define a default editor for re-editing in **$HOME/.mh_profile**. If an editor is not defined for re-editing in your **.mh_profile**, **whatnow** invokes the editor used in the previous editing session.
list [*flags*]	Displays the draft. For *flags*, you can specify any flag that is valid for the command serving as the *lproc*. (You can set a default **lproc:** entry in **$HOME/.mh_profile**.) If you specify any flags that are not valid for *lproc*, **whatnow** does not pass the path name of the draft to *lproc*.
push [*flags*]	Sends the message in the background. You can specify any valid flag for the **send** command.
quit [**-delete**]	Ends the **whatnow** session. If you specify **-delete**, **whatnow** deletes the draft. Otherwise, **whatnow** stores the draft.
refile [*flags*] +*folder*	Files the draft in the specified folder and supplies a new draft having the previously specified form. For *flags*, you can specify any flag that is valid for the command serving as the *fileproc*. (You can set a default **fileproc:** entry in **$HOME/.mh_profile**.)
send [*flags*]	Sends the message. You can specify any valid flags for the **send** command.
whom [*flags*]	Displays the addresses to which the message would be sent. You can specify any valid flags for the **whom** command.

Flags

-draftfolder +*folder*
Places the draft message in the specified folder. If you do not specify this flag, **whatnow** selects a default draft folder according to the information supplied in the MH profiles. You can define a default draft folder in **$HOME/.mh_profile**. If **-draftfolder** +*folder* is followed by *msg*, *msg* represents the **-draftmessage** attribute.

-draftmessage *msg*
Specifies the draft message. You can use one of the following message references as *msg*:

num	*sequence*	**first**
prev	**cur**	
next	**last**	

The default draft message is **cur**.

-editor *cmd*
Specifies that *cmd* is the initial editor for composing or revising the message. If you do not specify this flag, **whatnow** selects a default editor or suppresses the initial edit, according to the information supplied in the MH profiles. You can define a default initial editor in **$HOME/.mh_profile**.

-help
Displays help information for the command.

-nodraftfolder
Places the draft in the file *user_mh_directory*/**draft**.

-noedit
Suppresses the initial edit.

-prompt *string*
Uses *string* as the prompt. The default string is `What now?`

Profile Entries

Draft-Folder: Sets your default folder for drafts.
Editor: Sets your default initial editor.
fileproc: Specifies the program used to refile messages.
lasteditor-**next:** Specifies the editor used after exiting *lasteditor*.
lproc: Specifies the program used to list the contents of a message.
Path: Specifies your *user_mh_directory*.
sendproc: Specifies the program used to send messages.
whomproc: Specifies the program used to determine the users to whom a message would be sent.

Files

$HOME/.mh_profile	The MH user profile.
user_mh_directory/draft	The draft file.

Related Information

Other MH commands: "**comp**" on page 185, "**dist**" on page 336, "**forw**" on page 438, "**prompter**" on page 778, "**refile**" on page 817, "**repl**" on page 821, "**rmm**" on page 841, "**scan**" on page 871, "**send**" on page 893, "**whom**" on page 1222.

The **mh-alias**, **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference*.

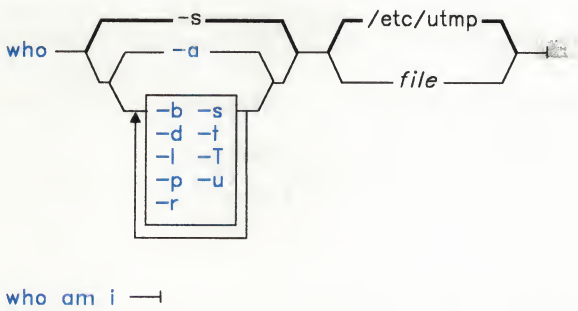
The "Overview of the Message Handling Package" in *Managing the AIX Operating System*.

who

Purpose

Identifies the users currently logged in.

Syntax



AJ2FL120

Description

The **who** command with no flags writes to standard output the login name, work station name, and date and time of login for all users currently on the system. Entering `who am i` displays your login name and work station name, and the date and time you logged on.

Note: The **who am i** command does not display the time of login when executed from a virtual terminal.

With flags, **who** can also display the elapsed time since line activity occurred, the process ID of the command interpreter (shell), logins, logoffs, restarts, and changes to the system clock, as well as other processes generated by the **init** process.

The general format of the output of **who** is as follows:

```
name [state] line time activity pid [location] [exit]
```

where:

- `name` is the user's login name.
- `state` indicates whether or not the line is readable by everyone (see the **-T** flag on page 1221).
- `line` is the name of the line as found in the directory **/dev**.
- `time` is the time the user logged in.

- `activity` is the hours and minutes since activity last occurred on that user's line. A dot (.) here indicates line activity within the last minute. If the line has been quiet more than 24 hours or has not been used since the last system startup, the entry is marked as old.
- `pid` is the process ID of the user's shell.
- `location` is the location associated with this line as found in file **/etc/ports**. This file can contain information about where the work station is located, the telephone number of the dataset, the type of a direct-connected work station, and other related information.
- `exit` is the exit status of ended processes (see the **-d** flag on page 1220).

To obtain information, **who** normally examines **/etc/utmp**. If you specify another *file*, **who** examines the named *file* instead. This *file* will usually be **/usr/adm/wtmp**, which contains the history of all logins since the file was last created or **/etc/.ilog**, which contains the history of invalid logins. Only someone operating with superuser authority or a member of the system group can examine **/etc/.ilog**

Note: This command only identifies users on the local node.

Flags

-a Processes **/etc/utmp** or the named *file* with all flags on.

-b Indicates the time and date of the most recent system startup. The **NLTIME** and **NLLDATE** environment variables control the format of the login time and date.

-d Displays all processes that have expired without being regenerated by **init**. The *exit* field appears for dead processes and contains the termination and exit values (as returned by **wait**) of the dead process. (This flag is useful for determining why a process ended.)

-l Lists only work stations not in use. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field doesn't appear.

-p Lists any active process that is currently active and has been previously generated by **init**.

-r Indicates the current **run-level** of the process.

-s Lists only the *name*, *line*, and *time* fields. (This is the default; thus, **who** and **who -s** are equivalent.) The **NLTIME** environment variable controls the format of the time.

-t Indicates the last change to the system clock by the superuser using the **date** command. The **NLTIME** environment variable controls the format of the time.

-T Displays the *state* of the work station line and indicates who can write to that work station as follows:

+ writable by anyone
- writable only by the superuser or its owner
? bad line encountered.

-u Displays the user name, work station name, login time, line activity, and process ID of each current user. The **NLTIME** environment variable controls the format of the login time.

Examples

1. To display information about who is using the system:

   ```
   who
   ```

   This lists the user name, work station name, and login time of all users currently using the system.

2. To display your user name:

   ```
   who am i
   ```

   This displays the user name you typed when you logged in, the name of the work station you are using, and the time you logged in. Your login user name may be different from your current user name if you have used the **su** command.

3. To display a history of logins, logoffs, system startups, and system shutdowns:

   ```
   who  /usr/adm/wtmp
   ```

Files

/etc/utmp
/usr/adm/wtmp
/etc/ports

Related Information

The following commands: "**date**" on page 281, "**init**" on page 521, "**login**" on page 584, "**mesg**" on page 642, and "**su**" on page 1026.

The **wait** system call and the **ports** and **utmp** files in *AIX Operating System Technical Reference*.

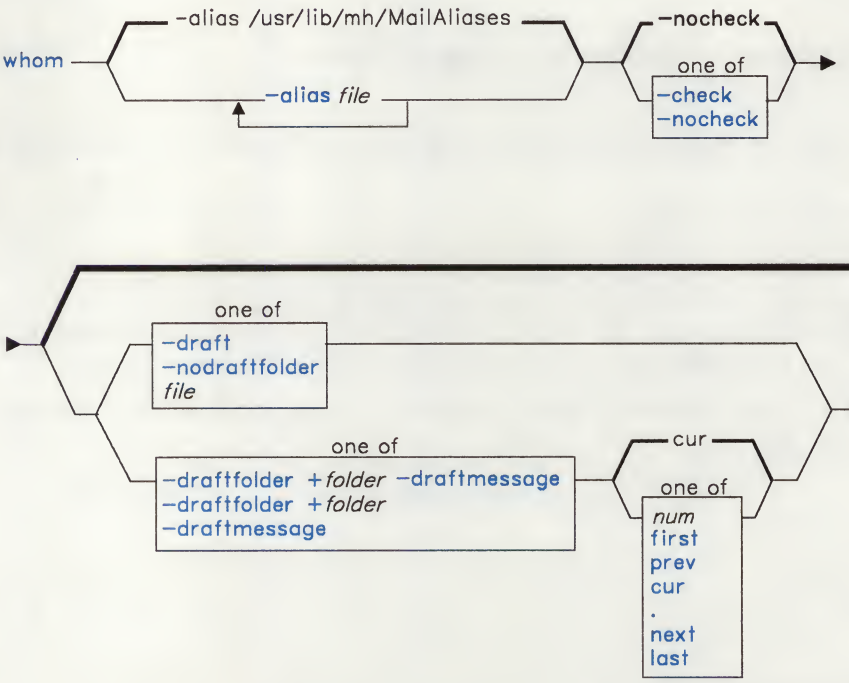
"Overview of International Character Support" in *Managing the AIX Operating System*.

whom

Purpose

Lists the addresses of the proposed recipients of a message and verifies the addresses.

Syntax



AJ2FL172

Description

The **whom** command is used to expand the headers of a message into a set of addresses. **whom** is also used to verify that those addresses are valid. **whom** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The message can reside in a draft folder or in a file. You can use one of the **-draft**, **-draftfolder**, **-draftmessage**, or **nodraftfolder** flags or the *file* argument to specify where the message resides.

If you want to verify the addresses, you must specify the **-check** flag.

Flags

-alias *file*
Specifies that *file* is a mail alias file to be searched for aliases. The default alias file is **/usr/lib/mh/MailAliases**.

-check
Checks to see if the addresses are valid.

-draft
Uses the header information in the file *user_mh_directory*/**draft** if it exists.

-draftfolder +*folder*
Uses the header information of the draft message in the specified folder. If you do not specify this flag, **whom** selects a default draft folder according to the information supplied in the MH profiles. You can define a default draft folder in **$HOME/.mh_profile**. If **-draftfolder** +*folder* is followed by *msg*, *msg* represents the **-draftmessage** attribute.

-draftmessage *msg*
Uses the header information in the specified draft message. You can use one of the following message references when specifying *msg*:

num	*sequence*	**first**
prev	**cur**	
next	**last**	

The default draft message is **cur**.

-help
Displays help information for the command.

-nocheck
Does not check to see if the addresses are valid. This is the default.

-nodraftfolder
Undoes the last occurrence of **-draftfolder** +*folder*.

Profile Entries

Draft-Folder: Sets your default folder for drafts.
postproc: Specifies the program used to post messages.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

Other MH commands: "**ali**" on page 48, "**post**" on page 758, "**whatnow**" on page 1215.

The **mh-alias**, **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference.*

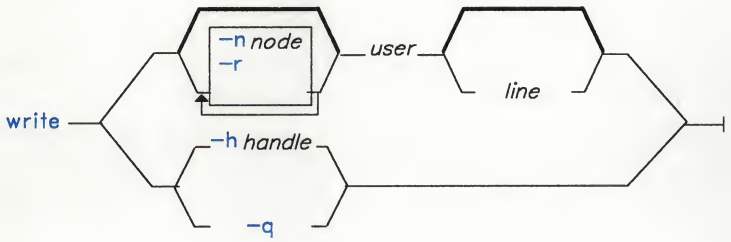
"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

write

Purpose

Sends messages to other users on the system.

Syntax



AJ2FL121

Description

A common use of this command is to ***converse*** with another logged-in *user*. That is, each user alternately sends and receives short messages from the other work station. Long messages can be sent by first putting the complete message in a file and then redirecting that file as input to the **write** command.

For another *user* to receive your message, that user must be logged in and must not have refused message permission. When a person you are trying to reach is not logged in, you get the message `user not logged in.` When the person you are trying to reach has refused message permission, you get the message `write: permission denied.`

When you run the **write** command, it immediately sends the following message, along with an attention-getting sound (the ASCII BEL character) to the person whose login name you entered.

`Message from` *yourid* `(tty`*nn*`)`
[*date*] . . .

After successful connection, **write** then sends two ASCII BEL characters to your work station to alert you that whatever you enter now is being sent to the other *user*. Sending continues until you press **Ctrl-D**, at which point **write** sends an end-of-text character to the other work station and exits.

At this point, the other *user* can respond by sending a **write** message back. For this type of exchange, the following convention is useful: When you first **write** to others, wait for them to **write** back before sending any text. End a message with a signal such as o (over) to alert the other person to reply. Use oo (over and out) when the conversation is finished.

When you **write** to a *user* logged in at more than one work station, **write** uses the first login instance found in file **/etc/utmp** as the message delivery point, and you get the message:

userid is logged on more than one place.
You are connected to "*work station*".
Other locations are:
work station

You can contact this *user* at another location by specifying the *line*. *line* indicates to which work station (**tty00**, for example) the message should be sent.

Permission to **write** to another *user* is granted or denied by the other *user* with the **mesg** command. Some commands deny message permission while they are running to prevent interference with their output. A user with superuser authority can **write** to any work station regardless of the work station's message permission.

If Distributed Services is installed on your system, you can use the **write** command to converse with users on other nodes. You can identify a user on a remote node explicitly by using the **-n** flag or implicitly through entries in the file **/etc/wwwmachines**. This file contains a list of node IDs or nicknames of machines at which a user may be contacted. If you use the **-n** flag, **write** does not run through the list of machines named in **/etc/wwwmachines**.

The **write** command is also used by **qdaemon** to send messages to users on other nodes and to wait for replies. The contents of the message becomes the reply. Certain keywords in the reply message are recognized as having special meaning. If the user replies with the word ok, then the original **write** exits with a status of 0. If the user replies with the word cancel, then the **write** exits with a status of 1. If the reply contains the word query, then the message associated with the handle given is displayed.

Flags

-n *node* Specifies a remote node. The *node* field may be a nickname or a node-ID.

-h *handle* Replies to a message sent by a utility or shell script using **write** with the reply option. The value to be used for *handle* is generated internally and supplied to the user in the text of the original message.

-r Generates a message handle, places it in the message header, sends the message, and waits for a reply. This flag is used by **qdaemon** for operator messages and can be put in shell scripts. It is not used for interactive conversations. An exit status of 0 indicates that the reply was `ok`, a status of 1 indicates that the reply was `cancel`, and an exit status of 2 indicates that the user could not be contacted.

-q Queries all messages awaiting replies from users on a node and displays them with their handles.

Examples

1. To write a message to a user who is logged in:

```
write scottie
I need to see you!  Meet me in the computer room at 12:30.
Ctrl-D
```

If your user ID is `kirk` and you are using work station `tty3`, `scottie`'s work station displays:

```
Message from kirk tty3...
I need to see you!  Meet me in the computer room at 12:30.
EOF
```

2. To hold a conversation:

```
write scottie
Meet me in the computer room at 12:30.
(o)
```

This starts the conversation. The `(o)` at the end stands for "over." It tells `scottie` that you are waiting for a response. *Do not* press **Ctrl-D** if you wish to continue.

Now `Scottie` replies by typing:

```
write kirk
I'm running tests at 12:30.  Can we meet at 3?
(o)
```

And you might respond:

```
write scottie
OK--the computer room at 3.
(oo)
```

The `(oo)` stands for "over and out," telling `Scottie` that you have nothing more to say. If `Scottie` is also finished (`oo`), then you both press **Ctrl-D** to end the conversation.

3. To write someone a prepared message:

```
write jay <message.text
```

This writes the contents of the file `message.text` to `fred`'s work station.

4. To write to the person using a certain work station:

```
write - console
The printer in building 998 has jammed.
Please send help.
Ctrl-D
```

This writes the message to the person logged in at the work station **/dev/console**.

5. To send a message to user `spuds` at node `partya`:

```
write -n partya spuds
Your new tape has just arrived,
come see me to pick it up.
Thanks!
Ctrl-D
```

6. Here is an example of a message sent by **qdaemon**:

```
Message from mary on node 10813661 [ Aug 17 10:03:34 ] ...
[ Sent by qdaemon, use "write -h 6398492" to reply ]

Please insert tape number 5 into rmt0.

EOF
```

To reply in the affirmative enter:

```
write -h 6398492
ok
Ctrl-D
```

To reply in the negative enter:

```
write -h 6398492
cancel
Ctrl-D
```

Note: With the **-h** flag, there is no need to supply the node ID or user ID. This information is tracked with the handle.

Files

/etc/utmp	Contains user and accounting information for the **who**, **write**, and **login** commands.
/etc/wwwmachines	Contains node IDs and nicknames of machines at which the users may be contacted.

Related Information

The following commands: "**mesg**" on page 642, "**nroff, troff**" on page 709, "**pr**" on page 761, "**sh**" on page 913, "**wall**" on page 1208, and "**who**" on page 1219.

writesrv

Purpose

Allows Distributed Services users to send messages to and receive messages from a remote system.

Syntax

AJ2FL254

Description

The **writesrv** command is a daemon that allows users in a Distributed Services environment to send messages to users on a remote system and receive responses from users on a remote system with the **write** command.

The **writesrv** utility receives incoming requests from a **write** command and creates a server process to handle the request. This server process communicates with the client process (**write**) and provides whatever services are requested.

To perform these services, the **writesrv** command creates an IPC queue using the key `0x203fe`. All requests for service are sent as messages to this queue. The server also creates an IPC queue profile named **IBMWRT***node* where *node* is the node ID of the local system. This profile may be examined with the **ipctable** command but should not be modified.

Related Information

The commands: "**write**" on page 1225 and "**ipctable**" on page 544.

The **msgget**, **msgctl**, **msgsnd**, and **msgrcv** system calls in *AIX Operating System Technical Reference.*

wump

Purpose

Plays the game Hunt the Wumpus.

Syntax

```
/usr/games/wump —⊣
```

OL805232

Description

A wumpus is a creature living in a cave with many rooms interconnected by tunnels. You move among the rooms trying to shoot the wumpus with an arrow and trying to avoid being eaten by the wumpus or falling into bottomless pits. There are also Super Bats that may pick you up and drop you in some randomly selected room. For moving among the rooms and shooting arrows, **wump** asks appropriate questions and follows your instructions.

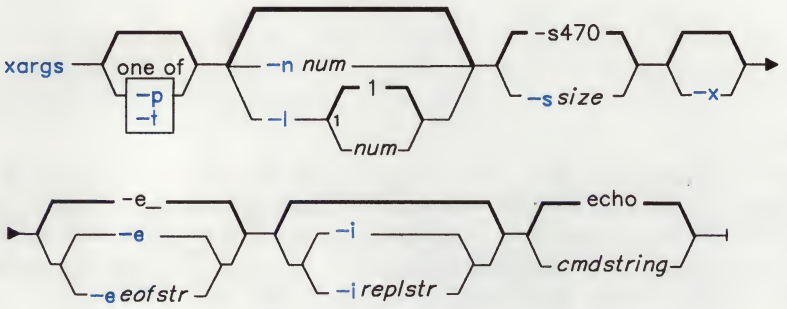
After either you kill the wumpus, the wumpus eats you, or you fall into a Bottomless Pit, **wump** asks if you want a new game. To quit the game at any time, press **INTERRUPT (Alt-Pause)** or **END OF FILE (Ctrl-D)**.

xargs

Purpose

Constructs argument lists and runs commands.

Syntax



OL805298

Description

The **xargs** command runs a command line. It constructs the command line by combining *cmdstring*, a string containing a command and its flags or parameters, with additional arguments read from standard input. It runs *cmdstring* as many times as necessary to process all input arguments. The default *cmdstring* is **echo**.

Arguments read from standard input are character strings delimited by one or more blanks, tabs, or new-line characters. You can embed a blank or a tab in arguments by preceding it with a \ (backslash) or by putting it in quotation marks. **xargs** reads characters enclosed in single or double quotation marks as literals and removes the delimiting quotes. It always discards empty lines.

Flags

-e[*eofstr*] Sets the logical end-of-file string to *eofstr*. **xargs** reads standard input until it encounters either an end-of-file character or the logical **EOF** string. If you do not specify the **-e** flag, the default *eofstr* is _ (the underline character). If you specify **-e** with no *eofstr*, **xargs** interprets the underline character as a literal character rather than as an end-of-file marker.

-i[*replstr*] Takes an entire line as a single argument and inserts it in each instance of *replstr* found in *cmdstring*. A maximum of five arguments in *cmdstring* can each contain one or more instances of *replstr*. **xargs** discards blanks and tabs at the beginning of each line. The argument constructed cannot be larger than 255 bytes. The default *replstr* is {}. This flag also turns on the **-x** flag.

-l[*num*] Runs *cmdstring* with the specified *num* of nonempty argument lines read from standard input. The last invocation of *cmdstring* can have fewer argument lines if fewer than *num* remain. A line ends with the first new-line character unless the last character of the line is a blank or a tab. A trailing blank or tab indicates a continuation through the next non-empty line. The default *num* is 1. This flag turns on the **-x** flag.

-n*num* Executes *cmdstring* using as many standard input arguments as possible, up to a maximum of *num*. The **xargs** command uses fewer arguments if their total size is greater than the number of bytes specified by the **-s***size* flag described below. It also uses fewer arguments for the last invocation if fewer than *num* arguments remain. When **-x** is present, each *num* argument must fit the *size* limitation specified by **-x**.

-p Asks whether or not to run *cmdstring*. It displays the constructed command line, followed by a ? . . . prompt. Press y to run the *cmdstring*. Any other response causes **xargs** to skip that particular invocation of *cmdstring*. You are asked about each invocation.

-s*size* Sets the maximum total size of each argument list. *size* must be a positive integer less than or equal to 470. The default *size* is 470 bytes. Note that the byte count for *size* includes one extra byte for each argument and the number of bytes in the command name.

-t Echoes the *cmdstring* and each constructed argument list to file descriptor 2 (usually standard error).

-x Stops running **xargs** if any argument list is greater than the number of bytes specified by the **-s***size*. This flag is turned on if you specify either the **-i** or **-l** flags. If you do not specify **-i**, **-l**, or **-n**, the total length of all arguments must be within the *size* limit.

Note: The **xargs** command ends if it cannot run *cmdstring* or if it receives a return code of -1. When *cmdstring* calls a shell procedure, the shell procedure should explicitly **exit** with an appropriate value to avoid accidentally returning -1. (See "**sh**" on page 913.)

Examples

1. To use a command on files whose names are listed in a file:

```
xargs  lint  -a  <cfiles1  *
```

If `cfiles` contains the text:

```
main.c  readit.c
gettoken.c
putobj.c
```

then **xargs** constructs and runs the command:

```
lint  -a  main.c  readit.c  gettoken.c  putobj.c
```

Each shell command line can be up to 470 bytes long. If `cfiles` contains more file names than fit on a single line, then **xargs** runs the **lint** command with the file names that fit. It then constructs and runs another **lint** command using the remaining file names. Depending on the names listed in `cfiles`, the commands might look like:

```
lint  -a  main.c  readit.c  gettoken.c . . .
lint  -a  getisx.c  getprp.c  getpid.c . . .
lint  -a  fltadd.c  fltmult.c  fltdiv.c . . .
```

This is not quite the same as running **lint** once with all the file names. The **lint** command checks cross-references between files. However, in this example it cannot check between `main.c` and `fltadd.c`, or between any two files listed on separate command lines.

For this reason you may want to run the command only if all the file names fit on one line. Tell **xargs** this by using the **-x** flag:

```
xargs  -x  lint  -a  <cfiles
```

If all the file names in `cfiles` do not fit on one command line, then **xargs** displays an error message.

2. To construct commands that contain a certain number of file names:

```
xargs  -t  -n2  diff  <<end
starting  chap1  concepts  chap2  writing
chap3
end
```

This constructs and runs **diff** commands that contain two file names each (**-n**2):

```
diff  starting  chap1
diff  concepts  chap2
diff  writing   chap3
```

The **-t** flag tells **xargs** to display each command before running it so that you can see what is happening. The `<<end` and `end` define a "Here Document," which uses the text entered before the `end` line as standard input for the **xargs** command. For more details, see "Inline Input Documents" on page 928.

3. To insert file names into the middle of commands:

```
ls | xargs -t -i mv {} {}.old
```

This renames all files in the current directory by adding `.old` to the end of each name. The **-i** tells **xargs** to insert each line of the **ls** directory listing where a {} appears. If the current directory contains the files `chap1`, `chap2`, and `chap3`, then this constructs the commands:

```
mv chap1 chap1.old
mv chap2 chap2.old
mv chap3 chap3.old
```

4. To run a command on files that you select individually:

```
ls | xargs -p -n1 ar r lib.a
```

This allows you to select files to add to the library `lib.a`. The **-p** flag tells **xargs** to display each **ar** command it constructs and ask if you want to run it. Type `y` and press **Enter** to run the command. Press **Enter** alone if you do not want to run it.

Related Information

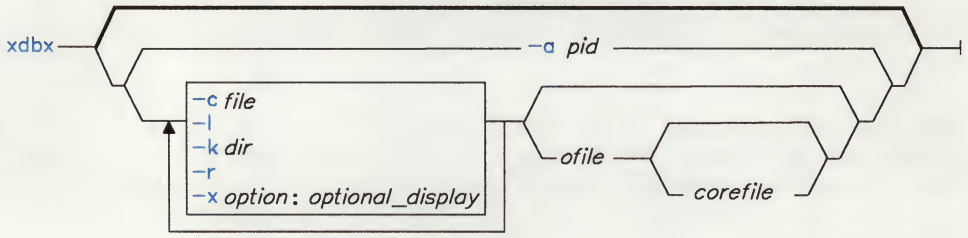
The following command: "**sh**" on page 913.

xdbx

Purpose

Provides an overlaying X-Window application for the dbx symbolic debugger.

Syntax



AJ2FL128

Description

The **xdbx** command is a user interface for the **dbx** source level debugger that provides a multi-window environment for interactive debugging of C, Pascal, and FORTRAN programs. All **dbx** flags, subcommands, and options can be used with **xdbx**.

Note: The **-x** flag is available with **xdbx** but not with **dbx**.

You must start X-Windows before using the **xdbx** command. The **xdbx** debugger displays up to three separate menu windows depending on the debugging mode. You select a debugging mode from **dbx** or **xdbx** using the **dbx** subcommand **set**.

Flags

-x *option*: **(***optional display***)** Runs the display portion of **xdbx** on a remote host. Where *option* is the name of the remote host, and the *optional display* is a small integer designating the display. The default is 0.

Related Information

The following command: "**dbx**" on page 284.

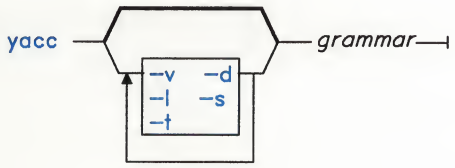
A discussion of how to debug programs in *AIX Operating System Programming Tools and Interfaces*.

yacc

Purpose

Generates a LR(1) parsing program from input consisting of a context-free grammar specification.

Syntax



OL805300

Description

The **yacc** command converts a context-free grammar into a set of tables for a simple automaton that executes an LR(1) parsing algorithm. The grammar can be ambiguous; specified precedence rules are used to break ambiguities.

You must compile the output file, **y.tab.c**, with a C Language compiler to produce a function **yyparse**. This function must be loaded with the lexical analyzer function **yylex**, as well as **main** and **yyerror**, an error-handling routine (you must provide these routines). The **lex** command is useful for creating lexical analyzers usable by **yacc**.

For more detailed discussion of **yacc** and its operations, see *AIX Operating System Programming Tools and Interfaces*.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-d Produces the file **y.tab.h**. This contains the **#define** statements that associate the **yacc**-assigned token codes with your token names. This allows source files other than **y.tab.c** to access the token codes by including this header file.

-l Does not include any **#line** constructs in **y.tab.c**. Use this only after the grammar and associated actions are fully debugged.

-s Breaks the **yyparse** function into several smaller functions. Since its size is somewhat proportional to that of the grammar, it is possible for **yyparse** to become too large to compile, optimize, or execute efficiently.

-t Compiles run-time debugging code. By default, this code is not included when **y.tab.c** is compiled. However, the run-time debugging code is under the control of **YYDEBUG**, a global variable for the **cc** command preprocessor. If **YYDEBUG** has a nonzero value, the C compiler (**cc**) includes the debugging code, whether or not the **-t** flag was used. Without compiling this code, **yyparse** will have a faster operating speed.

-v Prepares the file **y.output**. It contains a readable description of the parsing tables and a report on conflicts generated by grammar ambiguities.

Files

y.output	
y.tab.c	
y.tab.h	Definitions for token names.
yacc.tmp,	
yacc.debug	Temporary file.
yacc.acts	Temporary file.
/usr/lib/yaccpar	Parser prototype for C programs.

Related Information

The following command: "**lex**" on page 562.

The description of **yacc** in *AIX Operating System Programming Tools and Interfaces.*

ypbind

Purpose

Allows Yellow Pages client processes to communicate with the YP server.

Syntax

A5AC5011

Description

The **ypbind** commands remembers information that allows client processes on a single node to communicate with some **ypserv** process. The **ypbind** daemon must run on all machines using YP services, both YP servers and clients. The **ypbind** daemon is typically activated at system startup time from **/etc/rc.nfs**, if that file contains the appropriate entry.

The information handled by **ypbind** for a particular server is called a ***binding***. A binding associates a domain name with both the Internet address of a YP server and the server's port at which the **ypserv** process is listening for service requests.

When a request for an unbound domain comes in, the **ypbind** process broadcasts on the network in order to find a **ypserv** process that serves maps within that domain. When a domain is bound by a particular **ypbind** process, that binding is given to every client process on the node. The **ypwhich** command can be used to query the **ypbind** process on the local node or a remote node for the binding of a particular domain.

Bindings are verified before they are given out to a client process. If **ypbind** is unable to communicate with the **ypserv** process to which it is bound, it marks the domain as unbound. Then it informs the client process that the domain is unbound and tries again to bind the domain.

Requests for an unbound domain fail immediately. In general, a bound domain is marked as unbound when the node running **ypserv** crashes or is overloaded. Then **ypbind** attempts to bind to any YP server available on the network.

The **ypbind** daemon accepts requests to set its binding for a particular domain. Such requests are typically generated by the YP subsystem itself.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

File

/etc/rc.nfs

Related Information

The following commands: "**ypcat**" on page 1241, "**ypmatch**" on page 1245, "**yppush**" on page 1252, "**ypserv**" on page 1256, "**ypset**" on page 1254, and "**ypwhich**" on page 1258.

The section on Yellow Pages in *Managing the AIX Operating System.*

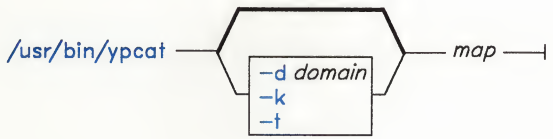
The Yellow Pages library section in *AIX Operating System Technical Reference.*

ypcat

Purpose

Displays values in a Yellow Pages data base.

Syntax



A5AC5016

Description

The **ypcat** command displays values in a Yellow Pages map specified by *map*, which can be either a map name or a map nickname. No YP server is specified since **ypcat** uses the YP network services.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-d Specifies a *domain* other than the default domain.

-k Displays the keys for those maps in which the values are null or the key is not part of the value.

-t Inhibits translation of *map* nickname to a map name. For example, the command sequence `ypcat -t passwd` fails because there is no map named *passwd*. However, the sequence `ypcat passwd` works because the map nickname `passwd` is translated to the map name `passwd.byname`.

-x Displays the map nickname table. The table lists valid nicknames and shows the map name associated with each nickname.

Related Information

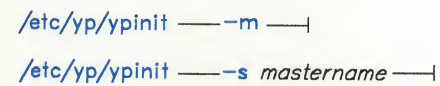
The following commands: "**ypmatch**" on page 1245 and "**domainname**" on page 340.

ypinit

Purpose

Builds and installs a Yellow Pages (YP) data base on the YP master server and YP slave servers.

Syntax



A5AC5003

Description

The **ypinit** command sets up a Yellow Pages data base on a YP master server or YP slave server. Only users with superuser authority can use **ypinit**.

With the **-m** option, **ypinit** initially sets up a YP master server which functions as the master for all maps in the data base. After initialization you can change the association of maps to masters. The **ypinit -m** command invokes the **make** procedure, which follows the instructions in **/etc/yp/Makefile**. By default, it uses the ASCII system files as input files to the data bases being created. See the Files section later in this discussion for a list of the default files **ypinit** uses to create the data base. The files used to build the data bases should be in their full-length form rather than in the abbreviated form used on client machines.

With the **-s** option, the YP data base on a YP slave server is set up by copying a data base from the YP server specified by the *masternamе* parameter (the machine's host name). The YP server from which the data base is being copied can be the YP master server, or a YP slave server whose data base is up to date and stable.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-m Specifies the local host as the YP master server, and installs the YP data base of maps.

-s *mastername* Sets up a YP slave server by installing a copy of the master data base from the host specified by the *mastername* argument.

Files

/etc/passwd
/etc/group
/etc/hosts
/etc/networks
/etc/protocols
/etc/netgroup (if set up)
/etc/rpc
/etc/services
/etc/ethers (if set up)
/etc/networks (if set up)
/usr/lib/aliases (if set up)

Related Information

The following commands: "**makedbm**" on page 632 and "**ypxfr**" on page 1260.

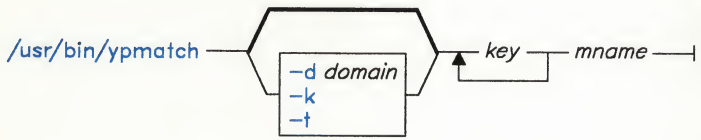
The Yellow Pages section in *Managing the AIX Operating System*.

ypmatch

Purpose

Displays the value of one or more keys from a Yellow Pages map.

Syntax



A5AC5018

Description

The **ypmatch** command displays the values associated with one or more keys from the Yellow Pages (YP) map specified by *mname*. The *mname* parameter may be either a mapname or a map nickname.

When multiple keys are specified by the *key* parameter, the same map is searched for all the specified keys. Pattern matching is not done for *keys*, so the exact string, including capitalization, must be given.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-d Specifies a *domain* other than the default domain.

-k Displays the key itself, followed by a colon, before displaying the value of the key. This flag is useful if the keys and their values are not identical, or if you specify so many keys that the output would be difficult to read.

-t Inhibits translation of *mname* to a map name. For example, the command sequence `ypmatch -t zippy passwd` fails because there is no map named *passwd*. However, the sequence `ypmatch zippy passwd` works because `passwd` is translated to the mapname `passwd.byname`.

-x Displays the map nickname table. The table lists valid nicknames and shows the mapname associated with each nickname.

Related Information

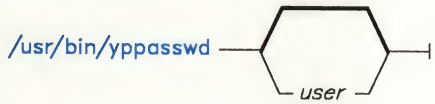
The following commands: "**domainname**" on page 340 and "**ypcat**" on page 1241.

yppasswd

Purpose

Establishes or changes your Yellow Pages password.

Syntax



A5AC5001

Description

The **yppasswd** command establishes or changes your Yellow Pages password entry. Your YP password may be different from the login password you use on your own machine.

When you enter the **yppasswd** command you are prompted for your old YP password. If you do not have an old password, press **Enter**. You then receive the prompt for your new password. If you do have an old YP password, you must type it in correctly in order to be prompted for a new password.

After entering your new password once, you are prompted for it a second time. The first and second entries must match in order for the new password to be validated.

Passwords should be at least four characters long, or six characters long if you use only upper case letters or only lower case letters. Superuser authority is required to change the YP password of another user.

Note: The YP password update protocol passes all the initial information to the server in a single RPC call. If you enter your old password incorrectly, you are not informed until after you enter your new password.

The **yppasswdd** daemon must be running on your YP server in order for your new password to take effect.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/yp/Makefile
/etc/passwd

Related Information

The following commands: "**passwd**" on page 735 and "**yppasswdd**" on page 1249.

The section on Yellow Pages in *Managing the AIX Operating System.*

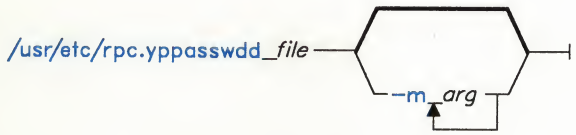
The **/etc/passwd** file format in *AIX Operating System Technical Reference.*

yppasswdd

Purpose

Handles password change requests from the **yppasswd** command.

Syntax



OL805511

Description

The **yppasswdd** daemon handles password change requests from the **yppasswd** command. It changes a Yellow Pages password entry in a file, provided the file has the same format as the **/etc/passwd** file. An entry in a file is changed only if the password presented by **yppasswd** matches the encrypted password of that entry.

The **-m** flag runs the **make** command using **Makefile** in the **/etc/yp** directory in order to update the changed password in the YP password data base. Any arguments that follow the **-m** flag are passed to the **make** command.

For example, if the Yellow Pages password file is named **/etc/yp/passwd**, then to have password changes take effect immediately, **yppasswdd** should be invoked as follows:

```
/usr/etc/rpc.yppasswdd /etc/yp/passwd -m passwd  \
  PASSWD=/etc/yp/passwd
```

The **yppasswdd** daemon must be run only on the YP master server for the YP password map. This daemon is not automatically invoked by **inetd** or by default like the other RPC daemons.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flag

-m Runs the **make** command using **Makefile** in the **/etc/yp** directory in order to update the changed password in the YP password data base. Any arguments that follow the flag are passed to the **make** command.

Files

/etc/yp/Makefile
/etc/passwd

Related Information

The following command: "**yppasswd**" on page 1247.

The section on Yellow Pages in *Managing the AIX Operating System.*

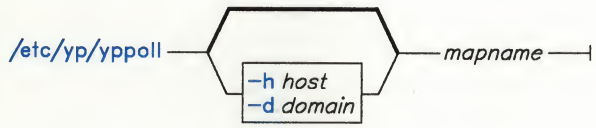
The **/etc/passwd** file format in *AIX Operating System Technical Reference.*

yppoll

Purpose

Displays the version of a Yellow Pages map located at a Yellow Pages server.

Syntax



A5AC5013

Description

The **yppoll** command queries the **ypserv** process for the order number and name of the host that is the YP master server for the map named by *mapname*.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-d Specifies a *domain* other than the default domain.

-h Queries the **ypserv** process at the specified *host* for the map parameters. If **host** is not specified, the YP server for the local host is used.

Related Information

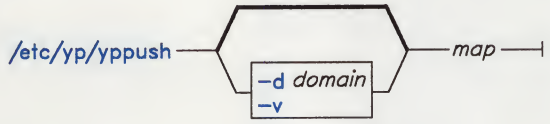
The following command: "**ypwhich**" on page 1258.

yppush

Purpose

Forces the distribution of a changed Yellow Pages map.

Syntax



A5AC5012

Description

The **yppush** command copies a new version of a Yellow Pages map from the YP master server to YP slave servers. It is normally run only on the YP master server from **/etc/yp/Makefile** after the data bases on the YP master server are changed.

The **yppush** command constructs a list of the YP server hosts by reading the YP map **ypservers** within the specified *domain*. The **ypservers** map contains the names of the machines which maintain YP maps. A request to transfer a map is sent to each YP server, along with the information needed to call back the **yppush** process.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-d Specifies a *domain* other than the default domain.

-v Verbose. Causes messages to be displayed when each server is called, and displays one message for each response. Only error messages are displayed if this flag is omitted.

Files

/etc/yp/*domainname*/ypservers.dir
/etc/yp/*domainname*/ypservers.pag

Related Information

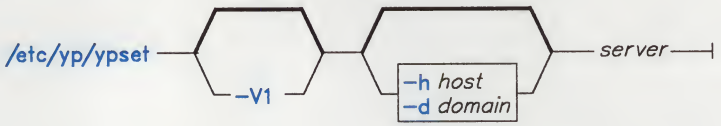
The following commands: "**makedbm**" on page 632 and "**ypxfr**" on page 1260.

ypset

Purpose

Directs the **ypbind** daemon to a particular server.

Syntax



A5AC5009

Description

The **ypset** command directs the **ypbind** daemon to get YP services for the specified *domain* from the **ypserv** daemon running on the specified *server*. The binding set by **ypset** is tested by **ypbind** when a client process tries to get a binding for the *domain*. If a binding is invalid (*server* is down or is not running **ypserv**), **ypbind** keeps trying to bind for the same domain.

The **ypset** command can be used to bind a client node that is not on a broadcast network, or to bind a client node on a broadcast network that is not running a YP host. The command can also be used for debugging Yellow Pages client applications, such as when a YP map only exists on a single YP server host.

If several hosts on the local network are supplying YP services, **ypbind** can rebind to another host while you are attempting to find out if the **ypset** operation succeeded. Then you might see the response `host2` after typing `ypset host1` and `ypwhich`. Such a response is simply a function of the the YP subsystem's attempt to balance its load among the available YP servers. The response occurs when *host1* is not running **ypserv** or is overloaded, and *host2* is running **ypserv** and is not overloaded.

The *server* parameter is the YP server to which a machine or node binds. It can be a name or an IP address. If a name is specified, **ypset** tries to use YP services to resolve the name to an IP address. The name can be resolved to an address only if the node has a current valid binding for the *domain*.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-d	Specifies a *domain* other than the default domain.
-h	Sets **ypbind** on *host* instead of locally.
-V1	Binds *server* for Version 1 YP protocol.

Related Information

The following commands: "**domainname**" on page 340, "**ypwhich**" on page 1258, "**ypserv**" on page 1256, and "**ypbind**" on page 1239.

ypserv

Purpose

Looks up information in the local data base of Yellow Pages maps.

Syntax

A5AC5010

Description

The **ypserv** daemon is typically activated at system startup time from **/etc/rc.nfs**, if that file contains the appropriate entry. The **ypserv** daemon runs only on Yellow Pages server machines that have a complete YP data base.

The **ypserv** daemon's primary function is to look up information in its local data base of YP maps. The operations performed by **ypserv** are defined for programmers by the **rpcsvc/yp_prot.h** header file, and for network implementors by the YP ***protocol specification***. Communication with **ypserv** is by means of Remote Procedure Calls.

There are four YP lookup operations that are performed on a specified map within some YP domain: **Match**, **Get_first**, **Get_next**, and **Get_all**. The **Match** operation takes a key and returns the associated value. The **Get_first** operation returns the first key-value pair from the map, and **Get_next** returns a certain number of the remaining key-value pairs (as specified in the program). The **Get_all** operation ships the entire YP map to a requestor in response to a single RPC request. These lookup operations are supplied as the following C-callable functions in **/lib/libc**: **yp_match**, **yp_first**, **yp_next**, and **yp_all**.

There are two other operations, **Get_order_number** and **Get_master_name**, that supply information about a map instead of map entries. Both order number and master name actually exist in the map as key-value pairs, but the server does not return either of them through the normal lookup functions. However, they will be visible if you examine the map with the **makedbm** command. **Get_order_number** and **Get_master_name** are supplied as the following C-callable functions in **/lib/libc**: **yp_order** and **yp_master**.

Log information is written to the file **/etc/yp/ypserv** if it exists when **ypserv** starts running.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/rc.nfs
/etc/yp/ypserv.log

Related Information

The following commands: "**ypbind**" on page 1239, "**ypcat**" on page 1241, "**ypmatch**" on page 1245, "**yppush**" on page 1252, "**ypset**" on page 1254, "**ypwhich**" on page 1258, and "**ypxfr**" on page 1260.

The section on the Yellow Pages in *Managing the AIX Operating System.*

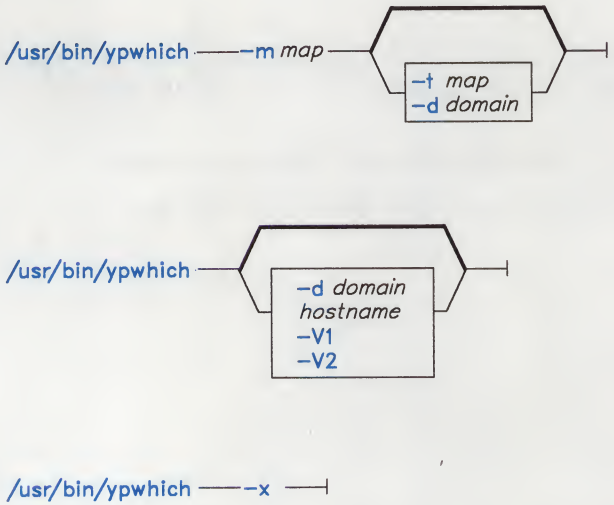
The Yellow Pages section in *AIX Operating System Technical Reference.*

ypwhich

Purpose

Displays the name of the host machine acting as the Yellow Pages server or as a YP map server.

Syntax



A5AC5008

Description

The **ypwhich** command shows which YP server supplies Yellow Pages services to a YP client, or shows which server is the YP master server for a map. With no flags or arguments specified, the **ypwhich** command displays the name of the YP server for the local machine. When a particular machine is specified by **hostname**, that machine is queried for the name of the YP server it is using.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-d	Specifies a *domain* other than the default domain.
-m	Finds the master YP server for a map. No *hostname* can be specified if **-m** is selected. The *map* parameter can be a mapname or a nickname for a map. A list of available maps is displayed when *map* is omitted.
-t	Inhibits nickname translation. This flag is used if *map* is identical to a nickname.
-V1	Displays name of server using Version 1 YP protocol.
-V2	Displays name of server using Version 2 YP protocol.
-x	Displays the map nickname table. The table lists valid nicknames and shows the mapname associated with each nickname.

Related Information

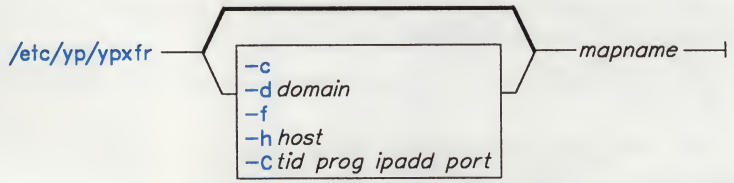
The following commands: "**ypset**" on page 1254 and "**yppoll**" on page 1251.

ypxfr

Purpose

Transfers a Yellow Pages map to the local host machine.

Syntax



A5AC5006

Description

The **ypxfr** command moves a Yellow Pages map to the local host by:

- Creating a temporary map in **/etc/yp/***domain* (which must already exist)
- Filling the map by enumerating its entries
- Fetching and loading the map parameters (order number and server)
- Deleting any old versions of the map
- Assigning *mapname* to the temporary map, making it the new map.

If used interactively, **ypxfr** sends output to the user's terminal. If invoked without a controlling terminal, **ypxfr** appends its output to the file **/etc/yp/ypxfr.log** (if the file exists). The **ypxfr** command is most often invoked from **crontab** or by the **ypserv** daemon.

To maintain consistent information between servers, **ypxfr** should be used periodically for every map in the YP data base. Some maps change more frequently than others, and therefore should be updated more often. For example, the services name map can change once every few months, whereas the mail aliases and password name maps can change several times a day.

You can use a **crontab** entry to perform periodic updates automatically. You can also group commands together in a shell script to update several maps at once. For useful examples, refer to **ypxfr_1perd**, **ypxfr_2perd**, and **ypxfr_1perh** in the **/etc/yp** directory.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-c Prevents a "clear current map request" from being sent to the local **ypserv** process. Use this flag if **ypserv** is not running locally at the time you are running **ypxfr**.

-C This option is *only* for use by **ypserv**. The parameters *tid*, *prog*, *ipadd*, and *port* (see syntax diagram) must be specified. The **ypserv** command specifies that **ypxfr** should call back a **yppush** process at the host with IP address *ipadd*, registered as program number *prog*, listening on *port*, and waiting for a response to transaction *tid*.

-d Specifies a *domain* other than the default domain.

-f Forces the transfer to occur even though the version at the server is not more recent than the local version.

-h Gets the map from *host* regardless of the server specified by the map. If *host* is not specified, **ypxfr** asks the YP service for the name of the server, and tries to get the map from there. The *host* parameter may be a name or an IP address of the form *a.b.c.d*.

Files

/etc/yp/ypxfr.log

Related Information

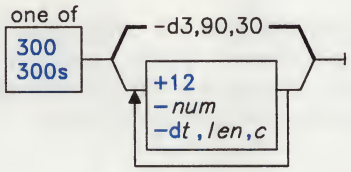
The following commands: "**crontab**" on page 222, "**ypinit**" on page 1243, "**makedbm**" on page 632, "**yppush**" on page 1252, and "**ypserv**" on page 1256.

300

Purpose

Handles special line-motion functions for DASI 300/300s work stations.

Syntax



OL805193

Description

Note: If your work station has a **PLOT** switch, make sure this switch is turned on before using this command.

The **300** command reads standard input, processes its input for printing on the DASI 300, GSI 300, or DTC 300 work stations, and writes to standard output. The **300s** command performs the same functions for the DASI 300s, GSI 300s, and DTC 300s. They convert the input files' motion control characters for half-line forward, half-line reverse, and full-line reverse into motion commands recognized by these work stations.

You can use the **300** and **300s** commands to draw Greek characters and other special symbols that require more than one vertical line, and it allows you to use 12-pitch text. For a discussion of special symbols and Greek characters supported by **300**, see "**greek**" on page 499.

The **nroff** command can be used with the **300** command to format text. **300** must be used if you use special delays or formatting options. You can either pipe from **nroff** to **300** or use the **-T300** flag with **nroff** to specify the printing device. The movement control of the **300** command usually produces better aligned output than **nroff -T300**.

When using **nroff**, the **-s** flag or **.rd** requests are required for inserting paper manually or changing fonts in the middle of a document. In these cases, you must press the line feed key to continue printing.

Using the **300** command with the **neqn** command will give you the best display of your equations. You can use the following sequence to display equations:

```
neqn file . . .  | nroff | 300
```

Note: Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from that position.

If your output contains Greek characters or reverse line feeds, use a friction-feed platen instead of a forms tractor. A forms tractor slips when reversing direction.

Flags

-d*t*,*len*,*c* Controls output delay factors. The default setting is **-d3,90,30**. DASI 300 is too slow to handle very long lines, too many tab characters, or long strings with no blanks and no identical characters. One null character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* nonblank, nontab characters. When a line is longer than *len* bytes, several nulls (the line length divided by 20, plus one) are inserted at the end of that line. In all three cases, the nulls delay the output enough to avoid a problem. Items can be omitted from the end of the list, implying the default values. Entering zero for *t* results in insertion of two null bytes per tab, while entering zero for *c* results in insertion of two null bytes per character.

When printing C Language programs, using `-d0,1` will help adjust for the many indentation levels. When printing files like **/etc/passwd**, using `-d3,30,5` will help print it properly.

This flag affects carriage return and line feed delays. The **stty** parameters **nl0 cr2** or **nl0 cr3** are recommended for most uses.

-num Controls the size of half-line spacing. The default half-line values (which are exact half-lines) of *num* are:

10-pitch, 6 lines-per-inch, num = 4
12-pitch, 8 lines-per-inch, num = 3
12-pitch, 6 lines-per-inch, num = 4

You can use other values for *num* to change the appearance of subscripts and superscripts. For example, **-2** makes **nroff** half-lines act like quarter-lines.

+12 Uses 12-pitch, 6 lines-per-inch text. The DASI 300 normally allows only two combinations: 10-pitch, 6 lines per inch, or 12-pitch, 8 lines per inch. To use the 12-pitch, 6 lines-per-inch combination, set the PITCH switch to 12 and use the **+12** flag on the command line.

Related Information

The following commands: "**450**" on page 1265, "**eqn**, **neqn**, **checkeq**" on page 395, "**graph**" on page 494, "**mesg**" on page 642, "**nroff, troff**" on page 709, "**stty**" on page 1018, "**tbl**" on page 1053, and "**tplot**" on page 1079.

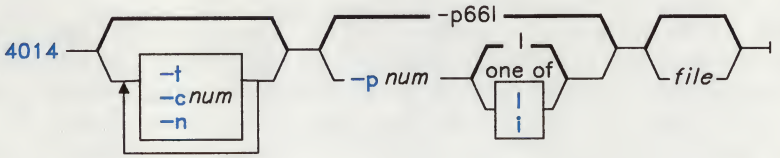
The **greek** miscellaneous facility in *AIX Operating System Technical Reference.*

4014

Purpose

Formats a full page 66-line screen display for a Tektronix 4014 work station.

Syntax



OL805195

Description

The **4014** command reads a *file* (standard input by default) and writes a 66-line page display to standard output. It also divides the screen into a specified number of columns, adding an eight-space page offset when it uses the default single-column format. It interprets tabs, spaces, backspaces, and TELETYPE Model 37 half-line and reverse-line sequences correctly. At the end of each page, **4014** waits for a line feed from the keyboard before continuing. While **4014** is waiting, you can send commands to the shell by entering !*AIX-cmd*, where *AIX-cmd* is a AIX command.

Flags

-c*num*	Divides the screen into *num* columns and waits after the last column. The default is a single, full page-width column.
-n	Starts displaying at the current cursor position and does not erase the screen.
-p*num***l** **-p***num***i**	Sets page length to *num* lines (**l**, the default) or to *num* inches (**i**).
-t	Does not wait between pages.

Related Information

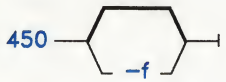
The following commands: "**pr**" on page 761, "**tc**" on page 1056, and "**troff**" on page 710.

450

Purpose

Handles special line-motion functions for the DASI 450 work station

Syntax



OL805194

Description

The **450** command reads standard input, processes its data for output on a DASI 450 or an equivalent work station (such as the DIABLO 1620 or XEROX 1700). It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions on standard output. It attempts to draw Greek characters and other special symbols in the same manner as the **300** command vertical line space. See "**greek**" on page 499 for a list of symbols supported by **450**.

Use **450** with the **nroff -s** flag or **.rd** requests when you need to insert paper manually or change fonts in the middle of a document. Instead of using the **Return** key in these cases, you must use the **LINE FEED** character to get any response. In many cases you can use **nroff -T450** instead of the **450** command. However, you must use **450** if you require special delays or options. In a few cases, using **450** may produce better aligned output. You can pipe the output of the **neqn** command to **450** to print equations neatly.

Notes:

1. Make sure the **PLOT** switch is turned on before using this command. Also, the **SPACING** switch should be in the desired position, either 10- or 12-pitch. For either setting, vertical spacing is 6 lines per inch unless changed to 8 lines per inch by an escape sequence.
2. Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from that position.
3. If your output contains Greek characters or reverse line feeds, use a friction-feed platen instead of a forms tractor. A forms tractor tends to slip when reversing direction.

Flag

-f Permits the use of ETX/ACK protocol with 1200 bps printers. You cannot use **450** with this flag in a pipeline or if you redirect its output. Instead it must drive the printer directly.

Related Information

The following commands: "**300**" on page 1262, "**eqn, neqn, checkeq**" on page 395, "**graph**" on page 494, "**greek**" on page 499, "**mesg**" on page 642, "**nroff, troff**" on page 709, "**stty**" on page 1018, "**tabs**" on page 1041, "**tbl**" on page 1053, "**tplot**" on page 1079, and "**troff**" on page 710.

The **greek** miscellaneous facility in *AIX Operating System Technical Reference.*